Immagine che contiene testo, schermata, Carattere, logo

Descrizione generata automaticamente

**MDDM**

Master’s Degree Program in

**Data-Driven Marketing**

**Improving Intrusion Detection Systems: Challenges with Public Datasets and the Role of Explainable AI**

A Practical Guide Using NFS-2023-TE and HIKARI-2021

Toscano Ludovico

Master Thesis

**NOVA Information Management School**

**Instituto Superior de Estatística e Gestão de Informação**

Universidade Nova de Lisboa

**NOVA Information Management School**

**Instituto Superior de Estatística e Gestão de Informação**  
Universidade Nova de Lisboa

**TITLE**

Subtitle

by

Toscano Ludovico

Master Thesis presented as partial requirement for obtaining the Master’s degree in Data-Driven Marketing, with a specialization in Data Science for Marketing.

**Supervised by**

Supervisor’s Fernando Bação. PhD, NOVA Information Management School.

July, 2024

# **Statement of Integrity**

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism, any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

*Rome, 27/06/2024*

Abstract

Intrusion detection systems based on public datasets deliver results only in academic papers but not in the real world because of flaws in how they are made. Should explainable algorithms be essential when building intrusion detection systems based on a machine learning model? A new paper highlights some of the deficiencies of these studies, but new datasets came out, and practical guides on tackling the problem are missing. Following the paper mentioned above, we have tried to see what improvements have been made and added new suggestions based on our findings. This work found issues on an existing dataset; moreover, we prove with explainable AI why building an intrusion detection system with these datasets should be avoided. Besides adding a contribution on what to avoid while making an intrusion detection system, we also show how some aspects of the process need to be deeply analyzed before proposing a new model.

**Keywords**

intrusion detection; explainable AI; NIDS; public dataset, HIKARI-2021, NFS-2023-TE, Cybersecurity

Table of Contents

[**Statement of Integrity** i](#_Toc171023497)

[Abstract ii](#_Toc171023498)

[List of Figures 2](#_Toc171023499)

[List of Tables 3](#_Toc171023500)

[**1.** **Introduction** 4](#_Toc171023501)

[**2.** **Background: Understanding Intrusion Detection Systems and Network Data** 6](#_Toc171023502)

[2.1. **What does an IDS do?** 6](#_Toc171023503)

[2.2. **What is an IP address, and what is a port?** 6](#_Toc171023504)

[2.3. **What is a flow?** 7](#_Toc171023505)

[2.4. **How these datasets are made** 8](#_Toc171023506)

[2.5. **What is a FIN and RST flag?** 9](#_Toc171023507)

[**3.** **Related work** 10](#_Toc171023508)

[**4.** **Methodology** 13](#_Toc171023509)

[4.1. **Dataset choice** 13](#_Toc171023510)

[4.2. **Explainable Algorithms in Machine Learning: SHAP and Beyond** 14](#_Toc171023511)

[4.3. **ML model choice** 19](#_Toc171023512)

[4.4. **Preprocessing** 19](#_Toc171023513)

[4.5. **Evaluation metrics** 19](#_Toc171023514)

[***5.*** ***Results*** 20](#_Toc171023515)

[*5.1.* ***Data set analysis*** 20](#_Toc171023516)

[5.2. **Model performance** 20](#_Toc171023517)

[*5.3.* ***Model explanations*** 23](#_Toc171023518)

[**6.** **Conclusion** 27](#_Toc171023519)

[**Bibliographical References** 28](#_Toc171023520)

[**Appendix A** 33](#_Toc171023521)

List of Figures

[Figure 5‑1 - NFS-2023-TE - decision tree feature importance with shap of bot class 19](#_Toc170409289)

[Figure 5‑2 - HIKARI-2021 - decision tree feature importance with shap of bruteforce-XML class 19](#_Toc170409290)

[Figure 5‑3- HIKARI-2021 - decision tree feature importance with shap of XMRIGCC CryptoMiner 20](#_Toc170409291)

[Figure 5‑4 - NFS-2023-TE - catboost impact on the model output of DDoS with shap sorted by highest magnitude of impact 21](#_Toc170409292)

List of Tables

[Table 3.1 - Related works \* shap was used but for feature selection \*\* they use all the features, but they group them (Panigrahi & Borah, 2018) 8](#_Toc170409372)

[Table 4.1- Typical dataset issues 10](#_Toc170409373)

[Table 5.1 - NFS-2023-TE and NFS-2023-nTe parameters 14](#_Toc170409374)

[Table 5.2 - NFS-2023-TE training times 15](#_Toc170409375)

[Table 5.3- NFS-2023-TE – average F1 score of 5-fold 15](#_Toc170409376)

[Table 5.4 - HIKARI 2021 parameters 16](#_Toc170409377)

[Table 5.5- HIKARI-2021 training times 17](#_Toc170409378)

[Table 5.6 - HIKARI-2021 F1 score and times 17](#_Toc170409379)

[Table 5.7 - fwd\_pkts\_payload.max quintiles 19](#_Toc170409380)

[Table 5.8 - Down\_up\_ration mean and std 19](#_Toc170409381)

# **Introduction**

In recent years, the global landscape of internet usage has undergone a dramatic transformation, with the number of users experiencing significant growth (Xiong et al., 2021). This expansion, driven by technological advancements and worldwide digitalization, has brought numerous benefits to society, including enhanced data sharing, network efficiency, and improved decision-making processes (Ahmetoglu et al., 2022).

However, this digital transformation has also introduced new challenges, particularly in cybersecurity. As the reliance on internet-connected systems grows, so does the potential for network threats and cyber-attacks (Walter et al., 2022). This trend has been further exacerbated by recent global events, such as the COVID-19 pandemic, which led to a surge in internet traffic due to increased remote work and stay-at-home activities. In 2022, global internet traffic reached a staggering 997,301 Gbps, a 30% increase from 2018 (TeleGeography, 2023).

The escalating threat landscape is evidenced by alarming statistics, such as a 29% increase in global cyber-attacks in 2021 (check point, 2021). This underscores the need for robust cybersecurity measures to protect individuals, businesses, and critical infrastructure.

In response to these growing threats, cybersecurity has evolved rapidly. Network Intrusion Detection Systems (NIDS) have become integral to organizational cybersecurity strategies. However, traditional signature-based and rule-based defensive mechanisms face challenges in coping with the increasing volume and complexity of information flowing through the internet (Gümüşbaş et al., 2021).

Researchers and computer engineers have turned to more advanced solutions to address these challenges, particularly in Artificial Intelligence (AI). Machine Learning (ML) and Deep Learning (DL) algorithms have shown promise in enhancing the performance of cyber defensive systems (Louk & Tama, 2023). These AI-based approaches offer the potential for more effective prediction and high-accuracy detection of emerging threats that may be difficult to identify using conventional antivirus programs.

While AI techniques have demonstrated impressive performance in various cybersecurity applications, such as intrusion detection, spam filtering, and malware identification (Sahakyan et al., 2021), a significant challenge lies in the trade-off between model performance and explainability. Interpretability and transparency have come to the forefront, particularly in light of regulations like the European Union's General Data Protection Regulation, which emphasizes the importance of understanding the logic behind AI-driven decisions (Goodman & Flaxman, 2017).

The field of Explainable AI (XAI) has emerged to address these concerns, offering strategies to make AI decisions more intelligible to humans. It has already been implemented in various domains, such as healthcare, business processes, financial and legal decisions, and autonomous vehicle (Nazar et al., 2021).

Building on the issue of explainability, this study practically addresses some specific criticisms raised by (Catillo et al., 2023). Starting from the premise that today, there are ML models that perform IDS work with incredible results, this study highlights how these academic results do not translate into real-world implementation of these models. The problem lies in a series of issues related to public datasets. Besides being a simplified version of reality, these datasets include errors in creating the dataset itself, leading to the easy achievement of good results by machine learning models. A dataset taken as an example in this study is CIC-IDS 2017 (Sharafaldin et al., 2018). The reason this dataset was chosen is that among the various datasets released in recent years, it is the most cited.

Starting from the criticisms developed by (Catillo et al., 2023), we will analyze two new datasets, NFS-2023-TE (Pekar & Jozsa, 2024) and HIKARI-2021 (Ferriyan et al., 2021). The choice of these datasets is not random, as NFS-2023-TE is the latest in a series of datasets created using the same data as CIC-IDS 2017, correcting some issues found in the original dataset. These issues have been addressed by WMTC-2021 (Engelen et al., 2021; L. Liu et al., 2022) and CRiSIS-2022 (Lanvin et al., 2023), leading to NFS-2023-TE, the latest in this series of improvements. Meanwhile, HIKARI-2021 is an entirely new dataset inspired by CIC-IDS 2017 that adds seven new requirements to the eleven used (Gharib et al., 2016) by CIC-IDS 2017.

The need to analyze new datasets instead of a more famous one arises for two reasons. The first is that NFS-2023-TE assumes that CIC-IDS 2017 contains errors, while the other most cited datasets are nowadays outdated. The second reason is that these datasets become obsolete quite quickly (Guerra et al., 2022), so it makes no sense to insist on working on old datasets.

This work aims to minimize the trade-off between explainability of these algorithms and performance. To achieve this result, we will demonstrate how even classical machine learning algorithms can achieve results comparable to deep learning models. After that, we will show how XAI algorithms like Shap (Lundberg & Lee, 2017) are fundamental in demonstrating the validity of a model or the dataset on which this model is based. We will then analyze the HIKARI-2021 dataset, which, to our knowledge, has never been done before. This analysis has led to the discovery of serious problems that make any model developed on these data unsuitable for real-world usage. This analysis will be conducted partly by verifying that the problems found in CIC-IDS 2017 are not present in HIKARI-2021, partly following what is exposed in (Catillo et al., 2023), and finally, use Shap as an XAI algorithm.

Consequently, this study does not want to be yet another attempt to obtain good results with a new model, as starting from flawed data would be a useless exercise. However, it wants to be practical work that brings together some issues already seen in the past to contribute to the drafting of some guidelines that can help develop better datasets in the future.

# **Background: Understanding Intrusion Detection Systems and Network Data**

This chapter briefly overviews key concepts in network security and intrusion detection systems (IDS). While not exhaustive, this introduction aims to equip readers with sufficient background knowledge to understand the context of our data science approach to IDS. The following sections offer concise explanations of fundamental ideas and processes in network monitoring and dataset creation for cybersecurity applications.

* 1. **What does an IDS do?**

An Intrusion Detection System (IDS) is a security tool that monitors network traffic for suspicious activities. Its main job is to:

1. Watch all network connections passing through a specific point, usually near the router.

2. Analyze this traffic to identify potential threats or attacks.

3. Alert security teams or automatically respond when it detects something suspicious.

Traditionally, many IDS systems used signature-based detection, which could only identify known attacks. However, according to recent research, machine learning (ML) models are increasingly being used to build more effective IDS. The critical advantage of ML-based IDS is their ability to understand the underlying patterns of attacks. This allows them to potentially detect new, previously unseen threats, unlike signature-based systems limited to recognizing only known attack patterns.

In our analysis, we will focus on how to evaluate the performance of ML models used in IDS. Specifically, we will explore:

1. Metrics for model accuracy: We will examine the advantages and limitations of using different evaluation metrics such as accuracy, precision, recall, and F1 score. This comparison will help us understand which metrics are most suitable for assessing IDS performance in different scenarios.

2. Inference time: We will evaluate the speed at which the model can make predictions. This is crucial for an IDS, as faster detection allows for quicker response to potential threats, minimizing the impact of an attack. For instance, in a data theft scenario, reducing detection time directly correlates with reducing the amount of data that could be stolen.

By focusing on these aspects, we aim to develop a comprehensive understanding of building and evaluating effective ML-based Intrusion Detection Systems that can adapt to new threats and provide robust protection against evolving cyber risks.

* 1. **What is an IP address, and what is a port?**

In the context of a dataset for an Intrusion Detection System (IDS), IP addresses and ports are represented as features. Each host (computer or device on a network) is identified by an IP address, which can be called its network "address". However, it is essential to note that a single host can have multiple active ports, which act like "doors" for different types of network traffic.

Ports, ranging from 0 to 65535, serve specific purposes. For example, port 80 is typically used for HTTP traffic, port 443 for HTTPS, and port 22 for SSH. This means one host can communicate using multiple services simultaneously, each through a different port.

In our dataset, we typically find four key pieces of information for each connection: the source IP address, source port, destination IP address, and destination port. While this information is crucial for network analysis, using it directly in machine learning model training can lead to problems.

The main issue is that IP addresses are often dynamically assigned and can change over time. They are also limited in number, especially with IPv4 (though IPv6 vastly expands this range). Similarly, while there are 65,536 possible ports, many are standardized for specific uses. Training a model on these specific values could create a bias where the model associates certain IPs or ports with malicious activity rather than learning the underlying patterns of the attack.

Attackers could exploit this bias. If they discover that the model considers specific IPs or ports as "safe," they could use these to bypass detection. Therefore, using these features for data correlation and initial analysis is generally better than training the detection model's core.

The primary reason for including IP addresses and ports in the dataset is to correlate records with the raw network traffic data, typically stored in pcap (packet capture) files. However, to make this correlation accurate, a timestamp is also necessary. The combination of IP, port, and timestamp uniquely identifies a specific network connection at a particular moment.

In summary, while IP addresses and ports are crucial for understanding network traffic, their direct use in machine learning models for cybersecurity can lead to vulnerabilities. Instead, they should be used carefully, primarily for data correlation and preprocessing, while the models should focus on more stable and generalizable features of network behavior.

## **What is a flow?**

A flow in these datasets typically represents a single TCP or UDP connection, recorded as one row in the dataset. Each flow is composed of multiple IP packets but is summarized into a single record with various features:

1. Identifier features: These include source and destination IP addresses, ports, and timestamps. They uniquely identify a connection.
2. Statistical features: These are derived from the connection data, such as flag counts, average packet size, maximum payload length, duration, etc. These features capture the behavior of the connection.

The number of features can vary significantly between datasets. Modern public datasets often have 80 or more features, with some new tools potentially offering up to 150 features.

There's a clear start (three-way handshake) and various defined endings for TCP connections. UDP, being connectionless, does not have standard open/close procedures. In datasets, UDP "connections" are typically defined as exchanges starting with the first observed packet and ending after a predefined timeout.

## **How these datasets are made**

Creating a dataset for training an IDS involves several key steps, building upon the concepts of IP addresses, ports, and flows discussed earlier:

* + 1. Testbed Setup:
* A network environment is created, including various devices like computers and servers.
* Some datasets may focus on specific environments, such as IoT networks, which can influence the protocols used.
  + 1. Traffic Generation:
* Normal traffic: Devices are programmed to simulate typical user behavior like web browsing.
* Attack traffic: Controlled attacks are performed within the network.
  + 1. Data Capture:
* Tools like tcpdump are used to capture network traffic during both typical and attack scenarios.
* The captured data is stored in pcap (packet capture) files containing detailed packet information.
  + 1. Flow Generation:
* Flowmeter tools such as CICFlowMeter (Lashkari et al., 2017) or NFStream (Aouini & Pekar, 2022) processes the pcap files.
* These tools aggregate individual packets into flows, creating the foundation for dataset samples.
  + 1. Feature Extraction:
* The flowmeter tools extract features from each flow, including statistical information like packet counts, flow duration, and flag counts.
* This process transforms raw packet data into a structured format suitable for machine learning.
  + 1. Labeling:
* Each flow is labeled as either normal or a specific attack type.
* Labeling is typically done by analyzing the traffic patterns and correlating them with the known attack times and targets.
  + 1. Dataset Compilation:
* The labeled flows are compiled into a CSV file, forming the final dataset.
* Each row in this file represents a single flow, with columns for various features and the label.
  + 1. Validation and Publication:
* The dataset undergoes validation to ensure accuracy and consistency.
* The processed CSV and original pcap files are often made available to researchers.
  + 1. Dataset Iterations:
* Some datasets, like WTMC-2021, CRiSIS-2022, and NFS-2023-TE, are created by reprocessing pcap files from CIC-IDS 2017.
* This approach allows improving and expanding existing datasets without capturing new network traffic.

## **What is a FIN and RST flag?**

IP packets, in addition to their payload, contain a series of metadata, including so-called flags. These flags perform various functions, including establishing or terminating the connection between two hosts. Specifically, to understand network traffic datasets, we need to know what the RST and FIN flags do.

There are three ways to close a TCP connection:

1. Exchanging a packet with the RST flag set
2. Exchanging two packets with the FIN flag set between the two hosts
3. A timeout if the connection is not used after a certain period

It is essential to know that when monitoring a TCP connection, we should theoretically see either one RST packet or two FIN packets passing through, but not more than one RST or one FIN. Among all the features used for today's datasets, this is one of the few that contains an absolute value that, if exceeded, indicates a malfunction during the dataset creation.

This behavior of RST and FIN flags provides a unique feature in network datasets. Unlike many other features with varying thresholds or patterns, the count of RST or FIN flags per connection has a clear, absolute limit. Exceeding this limit (more than one RST or more than two FIN packets per connection) strongly suggests either:

* An anomaly in the network behavior
* An error in the dataset creation process

For data scientists working with network traffic datasets, understanding these TCP flag behaviors can be crucial for:

* Data cleaning and preprocessing
* Feature engineering
* Anomaly detection
* Understanding the underlying network processes represented in the data

# **Related work**

* 1. **Introduction**

This section reviews recent literature on Intrusion Detection Systems (IDS) using machine learning (ML), focusing on studies that employ the HIKARI-2021 and CIC-IDS 2017 datasets. Our analysis aims to identify common trends and challenges in current IDS research.

* 1. **Paper Selection Criteria**

We selected six papers using HIKARI-2021 (the total available at the time of writing) and seven papers using CIC-IDS 2017 (two most cited papers using SHAP and five most cited overall). Some papers include additional datasets beyond these two.

* 1. **Evaluation Framework**

Our analysis utilizes criteria adapted from Catillo et al. (2023), focusing on ML implications for IDS. We excluded data partitioning issues as they were irrelevant to our non-sequential analysis and transferability due to dataset limitations. We added an explainability criterion for papers employing post hoc explainability algorithms. Table 3.1 summarizes our evaluation criteria and findings.

Table 3.1 - Related works \* shap was used, but for feature selection \*\* they use all the features, but they group them (Panigrahi & Borah, 2018)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Paper | *Avoid attack-revealing features and ease of detection* | *Avoid unmotivated complexity* | *Use of the evaluation metrics* | *Source code* | *Explainability* | *Updated dataset* | *All attacks* | *Dataset used* |
| (Kwon et al., 2023) | Yes | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Noori et al., 2023) | Yes | Yes | Yes | No | No | No | No | Hikari-2021 |
| (Louk & Tama, 2023) | - | No | Yes | No | No | No | No | Hikari-2021 |
| (Rajak et al., 2022) | - | No | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes & Lopes, 2022) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Fernandes et al., 2023) | No | Yes | Yes | No | No | No | Yes | Hikari-2021 |
| (Chauhan & Shah Heydari, 2020) | Yes | No | Yes | No | No \* | No | Yes | CIC-IDS 2017 |
| (Sarhan et al., 2022) | Yes | Yes | Yes | No | Yes | No | Yes | CIC-IDS 2017 |
| (Zavrak & Iskefiyeli, 2020) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| (Kurniabudi et al., 2020) | Yes | Yes | No | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (Yulianto et al., 2019) | Yes | Yes | Yes | No | No | No | No | CIC-IDS 2017 |
| (Maseer et al., 2021) | Yes | Yes | Yes | No | No | No | Yes\*\* | CIC-IDS 2017 |
| (J. Liu et al., 2021) | Yes | Yes | No | No | No | No | Yes | CIC-IDS 2017 |
| This work | Yes | Yes | Yes | Yes | Yes | Yes | Yes |  |

* 1. **Key Findings**
     1. Dataset Updates: No papers in our review used updated datasets. For HIKARI-2021, a new data capture addressing imbalance issues exists but has not been widely adopted. Similarly, the recent NFS-2023-TE, an improvement on CIC-IDS 2017, was too new for inclusion in the reviewed studies.
     2. Comprehensiveness of Attack Types: Some papers do not utilize all dataset categories, potentially inflating performance metrics.
     3. Attack-Revealing Features: In HIKARI-2021 studies, we identified an issue where the pandas-generated index inadvertently revealed attack patterns. Similarly, in NFS-2023-TE, the 'protocol' feature might introduce bias, as all attacks were performed over TCP.
     4. Model Complexity: Several papers propose complex deep learning algorithms without justifying this complexity or comparing them to simpler models.
     5. Comparative Analysis: Comparisons with previous work often lack data preparation or metric use consistency.
     6. Source Code Availability: The availability of source code varies among the reviewed papers, impacting reproducibility and further analysis.
  2. **Paper-by-Paper Analysis**
     1. (Kwon et al., 2023) - HIKARI-2021

This study tested various models, including Random Forest, XGBoost, MLP, and CNN. While comprehensive, only MLP and CNN were tuned, potentially biasing results. They innovatively tested for zero-day attack detection, finding MLP and CNN capable of detecting unseen brute force XML and probing attacks. However, they did not use the updated HIKARI-2021 dataset, missing out on improvements in class balance.

* + 1. (Noori et al., 2023) - HIKARI-2021

Focused on feature selection but failed to specify initial features used. They removed one attack class without explanation and used SMOTE for oversampling despite removing the smallest class. The paper contains inconsistencies in attack descriptions and visualizations, raising concerns about methodology transparency.

* + 1. (Louk & Tama, 2023) - HIKARI-2021

This recent paper lacks crucial details such as features used and whether classification was binary or multi-class. Despite claiming code availability, it was not provided upon request. Attempts to replicate their best model (bagging of 50 gradient boost machines) resulted in a model predicting everything as non-attack, highlighting reproducibility issues.

* + 1. (Rajak et al., 2022) - HIKARI-2021

Proposed a CNN-LSTM hybrid model. However, their feature selection process included both "traffic category" and "Label" as features, effectively including target information in the training data, which is a significant methodological flaw.

* + 1. (Fernandes et al., 2023; Fernandes & Lopes, 2022) - HIKARI-2021

These studies inadvertently used non-feature columns ("Unnamed: 0.1" and "Unnamed: 0") in their analysis. These columns, artifacts of data loading, allowed models (especially Random Forests) to learn attack patterns based on row numbers, introducing a serious bias.

* + 1. (Chauhan & Shah Heydari, 2020) - CIC-IDS 2017

Used SHAP for feature selection and employed GANs for data generation. However, they limited their study to DDoS attacks without justification, reducing the practical applicability of their findings.

* + 1. (Sarhan et al., 2022) - CIC-IDS 2017

Compared NetFlow and CICFlowMeter across datasets, concluding NetFlow-based datasets achieved better scores. However, this comparison is undermined by known issues with CICFlowMeter and potential labeling errors in the NetFlow version.

* + 1. (Zavrak & Iskefiyeli, 2020) - CIC-IDS 2017

Employed unsupervised models (Auto Encoder, Variation Auto Encoder, One-Class SVM) trained on normal traffic only. Their evaluation using AUC on an unbalanced dataset without providing weighted means limits the interpretability of their results.

* + 1. 9. (Kurniabudi et al., 2020) - CIC-IDS 2017

Used Information Gain for feature selection and compared with NSL-KDD. They innovatively regrouped attack classes to address imbalance, but this approach may mask differences between attack types. The study's use of only 20% of the dataset is a limitation.

* + 1. (Yulianto et al., 2019) - CIC-IDS 2017

Addressed imbalanced learning using AdaBoost with SMOTE, demonstrating improved oversampling and feature reduction results. However, their focus solely on DDoS attacks limits the broader applicability of their findings.

* + 1. (Maseer et al., 2021) - CIC-IDS 2017

Tested 48 model-hyperparameter combinations, focusing on multi-class classification with AIDS. They reduced classes to four, potentially oversimplifying the problem. Their comprehensive approach to model testing is commendable, but the class reduction may limit real-world applicability.

* + 1. (J. Liu et al., 2021) - CIC-IDS 2017

Compared SMOTE and ADASYN with LightGBM across multiple datasets. While they addressed class imbalance through stratified sampling, their limited use of evaluation metrics (only accuracy and false alarm rate) makes it difficult to fully assess model performance, especially given the imbalanced nature of the data.

This analysis reveals common themes across papers, including issues with dataset versions, feature selection, class imbalance handling, and limited or biased evaluation metrics. Many studies also suffer from a lack of code availability, hindering reproducibility.

# **Methodology**

## **Dataset choice**

In February 2024, the three datasets with the most citations in Google Scholars were CIC-IDS 2017 (Sharafaldin et al., 2018) with 3149 citations, NLS-KDD (Tavallaee et al., 2009) with 5131 citations, and UNSW-NB15 (Moustafa & Slay, 2015) with 2740 citations, these two are improvements based on the raw data of KDD Cup ’99 (Lee et al., 1999). We focus on two datasets derived from CIC-IDS 2017, illustrating why it may no longer be the best option.

By making CIC-IDS 2017, the authors have achieved for the first time the goal of covering all the 11 criteria given by (Gharib et al., 2016) by using CICFlowMeter (Lashkari et al., 2017). However, the analysis of the pcap files by (Engelen et al., 2021) and (L. Liu et al., 2022) revealed over 5% corruption in dataset labeling and non-compliance of CICFlowMeter with TCP connection closure standards (Brownlee et al., 1999). This led to the creation of WTMC-2021 and a patched version of CICFlowMeter. One year later, WTMC-2021 improved with the name of CRiSIS-2022 (Lanvin et al., 2023) by adding the attack port scan that was not labeled and sorting and removing the duplicates in the pcap file, again thanks to a new patch of CICFlowmeter. In 2023 (Pekar & Jozsa, 2024)noticed that even after the patching, CICFlowMeter was not closing the connection after a FIN or RST packet and was also making some missing or negative values, so they decided to make NFS-2023-TE by using NFStream (Aouini & Pekar, 2022). NFS-2023-TE creates a new flow at first FIN or RST flag like most of the flow analyzers do (Hofstede et al., 2014), while NFS-2023-nTE creates a new flow after a timeout like WTMC-2021 and CRiSIS-2022 do, this second dataset has been made only for the sake of comparison with its predecessor.

Things may change, however, since the authors of CICFlowMeter said in an email exchange that they are making a new tool written in Python with more than 50 new features for 130 features.

Hikari 2021 (Ferriyan et al., 2021) is a dataset inspired by CIC-IDS 2017 but includes seven new requirements; they also provide the pcap files, a csv file, and a pkl of the dataset. They offered 84 traffic features and two target features, one multi-categorical and the other one binary; among the 84 traffic features, most of them were inspired by the ones of CIC-IDS 2017. In this dataset, the authors have focused only on web attacks since 80% of the attacks nowadays are done on the application layer. So, they have performed three different attacks, but on the dataset, a fourth one is present since while they were analyzing the background traffic, they found a crypto miner attack. The raw data were extracted using tcpdump, while the labeling and analysis of the background traffic were done with Zeek, which also added some features that are not present in CIC-IDS 2017. These features are the IP address, source and destination ports, and uid. Since the paper's release, the dataset has received an update that improves the dataset by increasing the size of the minority classes.

We decided to build our models using NFS-2023-TE and HIKARI-2021, which, as far as we know, are the two most updated generic datasets. (Catillo et al., 2023) list some typical dataset issues in *Table 4.1*, we will see if NFS-2023-TE and HIKARI-2021 have improved compared to CIC-IDS 2017 mentioned in the paper.

Table 4.1- Typical dataset issues

|  |  |  |
| --- | --- | --- |
| Issue | HIKARI-2021 | NFS-2023-TE |
| *Simplification of the data collection environment* | Not addressed | Not addressed |
| *Contemporaneity and effectiveness of the attack* | Hikari 2021 focuses only on encrypted traffic at the application layer, saying that 80% of attacks are done at this level. As far as we know, this is the most updated dataset available; this would be a good option until a new one emerges. | The data used was generated seven years ago. Moreover, some attacks in the dataset have been proven ineffective nowadays. In (Catillo et al., 2021) they |
| *Representativeness of the normal baselines* | Not addressed | Not addressed |
| *Bugs of the feature extractor and incorrect ﬂow records* | Hikari 2021 does not provide any source code and mentions that the labeling was made with Zeek alongside an undefined Python tool, so we analyzed the dataset before using it. This part will be discussed later. However, in an email exchange with the authors, they said the source code would be released. | To avoid this problem, NFS-2023-TE was made with NFStream, which was chosen since it is an open-source tool with a broad user base, and the labeling process has been documented and released alongside the Python code used. |
| *Data Labeling (Was the traffic analyzed or labeled based on IP, port, and timestamp?)* | The background data were analyzed with Zeek, which led to the discovery of an attack. HIKARI 2021 does not provide the timestamp, and part of the payload is encrypted to ensure privacy; it is not possible to prove if there is any flaw in the labeling. | NFS-2023-TE should deliver on this point since it has been made on top of the refinement that has improved the original issues of the labeling part. |
| *Class imbalance* | Probing is around 2,9 times bigger than Bruteforce, which has 7988 samples. This difference is considerably smaller than NFS-2023-TE, making over-sampling possible if needed. | PortScan is around 14574 times bigger than Hearthbleed, with only 11 samples. With such a small sample, it is impossible to make a good model. |

* 1. **Explainable Algorithms in Machine Learning: SHAP and Beyond**

As models become increasingly complex in machine learning, the need for interpretability and explainability has grown significantly. This section explores critical approaches and tools developed to address this need, focusing on SHAP (SHapley Additive exPlanations) and related methodologies.

* + 1. **SHAP: A Game-Theoretic Approach to Explainability**

SHAP, introduced by (Lundberg & Lee, 2017), a Python library leverages game theory concepts to provide post hoc explanations for machine learning models. At its core, SHAP computes approximations of Shapley values, a concept from cooperative game theory used to fairly distribute each player's contribution (or, in this case, feature) to the game's outcome (or model prediction).

* + 1. **Understanding Shapley Values**

Shapley values are calculated by considering all possible combinations of features and assessing how each feature's presence or absence affects the model's output. This approach is considered "fair" because it accounts for all possible feature interactions and orders. For example, if we have features A, B, and C, we would consider scenarios like {A}, {B}, {C}, {A,B}, {A,C}, {B,C}, and {A,B,C} to determine each feature's contribution. However, computing exact Shapley values is computationally expensive, especially for models with many features. SHAP addresses this challenge by providing efficient approximation methods.

* + 1. **Key SHAP Explainers**

1. Kernel SHAP: A model-agnostic explainer adapted from LIME (Local Interpretable Model-agnostic Explanations) (Ribeiro et al., 2016). It creates a surrogate linear model to compute Shapley values and can be applied to any machine learning model.

2. Deep SHAP: A model-specific explainer for neural networks based on the DeepLIFT algorithm (Shrikumar et al., 2017).

3. Tree SHAP: Introduced in subsequent papers (Lundberg et al., 2018, 2020), this is a model-specific explainer for tree-based models and ensembles. It is notably faster than Kernel SHAP and Deep SHAP, often capable of explaining entire training sets in seconds.

4. Fast Tree SHAP: An optimized version of Tree SHAP (Yang, 2021), offering improved performance, especially in multi-core environments.

* + 1. **The Choice of SHAP for Explainability**

Our focus on SHAP, particularly TreeExplainer (for tree-based models), is motivated by its unique theoretical guarantees and practical efficiency. As stated by (Lundberg et al., 2020):

TreeExplainer offers several key advantages:

1. Fast Local Explanations: It rapidly computes local explanations, which is crucial for real-time or large-scale applications like intrusion detection systems.

2. Guaranteed Consistency: The explanations maintain consistency across different model outputs, ensuring reliability in interpretation.

3. Polynomial Time Complexity: TreeExplainer reduces the complexity of exact Shapley value computation from exponential to polynomial time, making it feasible for practical use on large datasets and complex models.

4. Theoretical Soundness: Within the class of additive feature attribution methods, Shapley values (as computed by SHAP) are the only approach that satisfies three critical properties:

a) Local Accuracy (Additivity): The explanation's attribution values sum up to the model's output for a specific input, ensuring that the explanation accurately reflects the model's behavior for that instance.

b) Consistency (Monotonicity): If a model changes so that a feature's contribution increases or stays the same regardless of other inputs, that feature's attribution will not decrease. This property ensures that the explanations reflect genuine changes in the model's logic.

c) Missingness: This property ensures that features that do not contribute to a prediction are assigned zero importance.

These properties make SHAP, particularly TreeExplainer for tree-based models, a theoretically sound and practically efficient choice for explaining complex models in intrusion detection systems. The combination of fast computation and guaranteed consistency is precious in security contexts where both speed and reliability of explanations are crucial.

* + 1. **Model-Agnostic vs. Model-Specific Explainers**

Kernel SHAP is model-agnostic, meaning it can work with any machine-learning model. This flexibility comes at the cost of computational efficiency and the need for carefully chosen background data. In contrast, Tree SHAP and Deep SHAP are model-specific, designed to leverage the particular structures of tree-based models and neural networks. This specialization allows for more efficient computations and often more accurate explanations but limits their applicability to specific model types.

* + 1. **True-to-Model vs. True-to-Data Explanations**

An important distinction in explainable AI is between true-to-model and true-to-data explanations (H. Chen et al., 2020):

* True-to-Model Explanations: Provided by Tree SHAP and Linear SHAP, these explanations assign non-zero importance only to features actually used by the model. They break feature dependencies following causal inference principles (Janzing, 2019).
* True-to-Data Explanations: Other explainers provide these, which may assign equal importance to highly correlated features, even if the model only uses one.
  + 1. **Understanding SHAP Mechanics and Perturbation Approaches**

SHAP uses different approaches to assess feature importance, depending on the specific explainer and model type:

1. Kernel SHAP:

* Uses perturbation with background data.
* Background Data: A subset of the dataset used as a reference point.
* Perturbation: Randomly swaps feature values between the explained instance and the background data.
* Accuracy depends on:
  + The size of the background data (typically 100-1000 observations)
  + The number of perturbations per observation
  + For global explanations, the size of the sample being explained
* It's crucial to use realistic background data (e.g., sampled from the dataset or k-means centroids) to avoid skewing the explanations.

2. Tree SHAP:

* Can use tree path-dependent perturbation.
* Does not require separate background data.
* Instead, it follows the nodes of the tree, which contain all the training data.
* This approach allows for efficient computation by leveraging the structure of tree-based models.

3. Linear SHAP:

* Can use interventional feature perturbation.
* Does not require background data.
* Instead, it reads the linear model weights to provide explanations.
* This approach allows for true-to-model explanations by directly interpreting the model's coefficients.
  + 1. **Computational Complexity**

The computational efficiency of SHAP methods varies significantly. Kernel SHAP, being model-agnostic, is the most computationally expensive, especially for high-dimensional data or large datasets. By leveraging the structure of tree-based models, Tree SHAP achieves much faster computation times, often orders of magnitude faster than Kernel SHAP for the same dataset. Linear SHAP is typically the fastest, using the model coefficients directly.

* + 1. **Importance of Background Data**

For Kernel SHAP, the choice of background data is crucial. It serves as a reference point against which feature contributions are measured. Poor selection of background data can lead to misleading explanations. Ideally, background data should represent the features' meaningful "average" or baseline state.

* + 1. **Types of SHAP Explanations**

1. Local Explanations: Explain individual predictions. For example, in an intrusion detection system, a local explanation might show which network traffic features most contributed to classifying a particular connection as malicious.

2. Global Explanations: Provide overall feature importance across the dataset. These can be computed using either the mean absolute SHAP value or the maximum absolute SHAP value across observations. Global explanations in an IDS context might reveal which features are most important for detecting intrusions across all traffic.

* + 1. **Relationship between Local and Global Explanations**

Global explanations are typically derived by aggregating local explanations across many instances. This aggregation can be done by taking the mean or maximum of absolute SHAP values. While global explanations provide an overview of feature importance, they may obscure nuances captured in local explanations, especially for models with complex decision boundaries.

* + 1. **Limitations of SHAP**
* Computational Cost: Especially for Kernel SHAP with large datasets or high-dimensional features.
* Handling of Correlated Features: Kernel SHAP may distribute importance among correlated features in ways that can be counterintuitive.
* Assumption of Feature Independence: Some SHAP methods assume feature independence, which may not be in real-world datasets.
  + 1. **Post Hoc Explanations**

Post hoc explanations refer to interpretability methods applied after a model has been trained. These techniques aim to explain the decisions or predictions of a model without altering its internal structure or training process. SHAP is a prime example of a post hoc explanation method. The key characteristics of post hoc explanations include:

* Applied After Training: They are used to interpret a model's behavior after it has been trained and without modifying it.
* Can Be Model-Specific or Model-Agnostic: Some post hoc methods are designed for specific types of models (like Tree SHAP for tree-based models), while others can be applied to any model (like Kernel SHAP).
* Separate Process: After model training and prediction, the explanation is generated as a separate step.
* No Impact on Model Performance: They do not affect the model's predictive capabilities.
* Flexibility: Can be applied selectively to specific predictions or instances of interest.

While post hoc methods like SHAP offer powerful explanatory capabilities, they do add an additional computational step after model training and prediction. This contrasts inherently interpretable models, which provide explanations as an integral part of their structure or training process.

* + 1. **Alternative Approaches: Glass Box Models**

In contrast to models requiring post hoc explanations, we also explored alternative approaches that offer inherent explainability, specifically the Explainable Boosting Machine (EBM) (Lou et al., 2013). EBM, a Generalized Additive Model with Interactions (GA2M), represents a significant advancement in interpretable machine learning.

As stated by the authors of EBM, this model aims to achieve predictive power comparable to complex deep learning models while providing clear, interpretable explanations similar to simpler glass box models like linear regression. This combination of high performance and inherent interpretability makes EBM particularly interesting for applications where accuracy and explainability are crucial, such as intrusion detection systems.

Our study included EBM as a glass box model for comparison. Our approach focused on comparing the training time of EBM with other models. This comparison is particularly relevant because traditional black box models require additional time to run SHAP to generate explanations after training. In contrast, EBM provides explanations inherently during the training process, potentially offering a more time-efficient solution for scenarios where both model performance and interpretability are crucial.

By including EBM in our analysis, we aimed to evaluate its predictive performance and efficiency in providing interpretable results. This comparison allows us to assess the trade-offs between post hoc explainability methods like SHAP and inherently interpretable models like EBM in the context of intrusion detection systems, considering both predictive power and the time required to obtain explanations.

* 1. **Model Selection and Rationale**

Our choice of models for this study was driven by several key factors, including the unique advantages of Tree SHAP, the performance of boosted models on imbalanced datasets, and the need to balance explainability with performance. This approach aligns with future improvements highlighted in the survey by (Zhang et al., 2022) on explainable AI for cybersecurity, particularly addressing the trade-off between explainability and performance.

* + 1. **Importance of Tree SHAP**

Tree SHAP emerged as a crucial factor in our model selection. Its ability to provide fast, consistent, and theoretically sound explanations for tree-based models made it an ideal choice for our explainability needs. Most importantly, Tree SHAP offers true-to-the-model explanations, meaning they directly reflect the model's decision-making process rather than approximating it. This feature is critical for understanding how our models function, especially in the sensitive domain of intrusion detection.

* + 1. **Focus on Boosted Models**

Nearly all the models we selected, except for the Decision Tree, are boosted. This choice was deliberate and essential, especially given the imbalanced nature of our datasets. Boosted models, such as Random Forest, LightGBM, XGBoost, and CatBoost, have demonstrated superior performance on imbalanced datasets. This is crucial for our case with NFS-2023-TE, where some categories have tiny sample sizes. Combining these boosted models with Tree SHAP allows us to leverage their high performance while maintaining true-to-the-model explainability.

* + 1. **Handling Imbalanced Data**

The imbalanced nature of our datasets significantly influenced our model selection. While we considered deep learning models, they struggled with the imbalanced data. Moreover, the additional complexity of explaining deep learning models and the increased explanation time made them less suitable for our study. We decided against using oversampling techniques to maintain the integrity of the original data distribution.

* + 1. **Balancing Simplicity and Performance**

We included simpler models like Decision Trees to provide a baseline for comparison. This allows us to assess the trade-off between model complexity and performance. The Decision Tree, combined with Tree SHAP, also offers insights into the dataset structure and model behavior, with explanations that are particularly easy to interpret due to the model's simplicity.

* + 1. **Exploration of Other Classic Models**

We tested other classic machine learning models such as SVM, Logistic Regression, and KNN. However, these models did not provide satisfactory results on our datasets, further justifying our focus on tree-based and boosted models.

* + 1. **Inclusion of Explainable Boosting Machine (EBM)**

The EBM model was included as it promises to deliver performance comparable to complex models while maintaining inherent explainability. This aligns with our goal of balancing performance and explainability and provides an interesting contrast to the post-hoc explanations of Tree SHAP.

* + 1. **Selected Models**

Based on these considerations, we chose the following models for our study:

1. Decision Tree (DT): As a baseline model and for its inherent interpretability.

2. Random Forest (Breiman, 2001): A powerful ensemble method known for its robustness.

3. LightGBM (Ke et al., 2017): An efficient gradient boosting framework.

4. XGBoost (T. Chen & Guestrin, 2016): Another popular and highly effective boosting algorithm.

5. CatBoost (Dorogush et al., 2017): Known for its performance and handling of categorical features.

6. Explainable Boosting Machine: For its balance of performance and inherent explainability.

These models, coupled with Tree SHAP for explanation (except for EBM, which provides inherent explanations), allow us to thoroughly explore the trade-off between model performance and explainability in intrusion detection systems. The true-to-the-model nature of Tree SHAP explanations ensures that we can trust the insights gained from our models, which is crucial in a security-critical domain like intrusion detection. This approach enables us to address the challenges of imbalanced datasets while providing meaningful, accurate, and efficient explanations, crucial for the practical application of machine learning in cybersecurity.

* 1. **Data Preprocessing and Sampling Methodology**

To address the challenge of imbalanced datasets and improve our model performance, we implemented a careful under-sampling strategy for both the HIKARI-2021 and NFS-2023-TE datasets. This approach was chosen to balance the classes while maintaining the integrity of the original data distribution.

* + 1. **HIKARI-2021 Dataset**

1. Under-sampling: We reduced all classes to 7,988 samples each. This number was chosen based on the size of the smallest class in the dataset.

2. Class Merging: We merged the Benign and Background traffic classes to create a single non-malicious category. This simplification helps in focusing on distinguishing between normal and malicious traffic.

* + 1. **NFS-2023-TE Dataset**

1. Under-sampling: We reduced most classes to 738 samples each. This number was determined through experimentation as the optimal balance between class representation and overall model performance.

2. Exception for Smaller Classes: Classes with fewer than 738 samples were left unchanged to preserve all available information for rare attack types.

* + 1. **Limitations and Considerations**

1. Data Loss: We acknowledge that under-sampling results in a significant reduction of data, which is a limitation of this study. However, we deemed this preferable to removing entire attack classes, which would have limited the model's ability to detect a full range of attacks.

2. Oversampling Rejection: Given the extreme imbalance in NFS-2023-TE (with the smallest class having only 11 samples), we decided against oversampling. This decision was made to avoid the risk of overfitting on synthetic data, which could lead to unreliable model performance on real-world data.

* + 1. **Training and Validation Methodology**

1. Stratified Split: We used 80% of the under-sampled data for training, ensuring that the class distribution was maintained in training and test sets.

2. Cross-Validation: We implemented 5-fold cross-validation to ensure robust model evaluation and mitigate the overfitting risk.

This methodology represents a balanced approach to handling the challenges presented by our imbalanced datasets. While it involves some compromise regarding data utilization, it allows us to train models that can recognize a full spectrum of attack types while maintaining a balanced view of the problem space. The use of cross-validation further strengthens the reliability of our results, helping to ensure that our models' performance is consistent across different subsets of the data.

* 1. **Evaluation Metrics and Handling Class Imbalance in Multi-class NIDS**

In Network Intrusion Detection Systems (NIDS), choosing appropriate evaluation metrics is crucial, especially when dealing with multi-class, imbalanced datasets. Our approach addresses several key considerations:

* + 1. **Binary vs. Multi-class Classification**
* Some studies simplify NIDS to binary classification (attack vs. benign), where standard metrics like F1 score are straightforward to interpret.
* Our study focuses on multi-class classification to distinguish between different attack types, providing more detailed insights but requiring a more nuanced evaluation approach.
  + 1. **Limitations of Common Metrics**

Accuracy is overly simplistic when evaluating NIDS performance, as it can be misleadingly high in imbalanced datasets even when the model fails to detect rare attacks. While precision and recall provide more insight, they are still insufficient when used individually; precision may be high but recall low (or vice versa), failing to capture the overall effectiveness. Therefore, we opted for the F1 score, which balances precision and recall, providing a more comprehensive measure of the model's performance across common and rare attack classes..

* + 1. **Importance of Detecting All Attack Types**
* We emphasize the critical need to detect all types of attacks, including those with very few instances (e.g., attacks with only 11 samples in NFS-2023-TE).
* This aligns with real-world security requirements where missing a rare attack could have severe consequences.
  + 1. **F1 Macro and F1 Weighted formula**

Where N is the number of classes, i represents a specific class, and wi is the weight of a class in the dataset.

* + 1. **Comparison of F1 Macro and F1 Weighted**

1. F1 Macro:

* Calculates the arithmetic mean of F1 scores for each class.
* Treats all classes equally, regardless of their frequency in the dataset.
* More sensitive to the performance of minority classes.

2. F1 Weighted:

* Calculates a weighted average of F1 scores, where weights are proportional to class frequencies.
* Classes with more samples have a higher impact on the final score.
* Reflects performance with respect to the dataset's class distribution.
  + 1. **Key Insights from F1 Macro and F1 Weighted**

1. Imbalanced Dataset Sensitivity:

* F1 macro is more indicative of performance on rare attacks.
* F1 weighted might show high scores even if rare attacks are misclassified, provided common attacks are correctly identified.

2. Use Case in Our Study:

* For HIKARI 2021 (after under-sampling): These metrics will yield identical results as the dataset becomes balanced.
* For NFS-2023-TE (remains imbalanced):

a) A high F1 weighted with a lower F1 macro would indicate good performance on common attacks but potential struggles with rare ones.

b) Similar F1 macro and weighted scores suggest consistent performance across all classes.

3. Overall Model Assessment:

* F1 macro provides a more stringent evaluation across all attack types.
* F1 weighted reflects performance experienced in a real-world scenario with the given class distribution.

4. Improvement Focus:

* A significant difference between scores can guide improvement efforts:
  + Lower F1 macro suggests the need to improve the detection of minority classes.
  + Low scores in both indicate overall performance issues.

By employing this comprehensive evaluation strategy, we aim to provide a nuanced and accurate assessment of our models' performance across all attack types. This approach helps avoid the pitfalls of oversimplified metrics. It ensures that our evaluation aligns with real-world requirements of intrusion detection systems, where the ability to detect every type of attack, no matter how rare, is crucial for maintaining network security. The combination of F1 macro and F1 weighted, along with supplementary analyses, allows us to balance the assessment of overall effectiveness in real-world scenarios with the critical requirement of detecting all types of network intrusions, regardless of their frequency.

# ***Results***

* 1. ***Data set analysis***

While in NFS-2023-TE, we did not find any flaw in the dataset, in HIKARI 2021, we noticed that the flag count for FIN and RST is higher than how it is supposed to be, having a maximum of 140 FIN packages and 110 RST packages. Moreover, the most extended flow lasted 17942 seconds, about 4.9 hours. Since the capturing sessions of HIKARI 2021 lasted 3 to 5 hours, the tool did not have a timeout, nor was it closing the connection at the first RST or FIN flag. Likely, the tool was always ignoring any timeout, RST, or FIN packet since if we filter for all the connections that lasted less than 181 seconds, with less than 2 RST and 3 FIN packets, we obtain only 104 samples for bruteforce and 59 for bruteforce-XML. For instance, NFS-2023-TE closes a connection after 120 seconds, and CICFlowMeter was supposed to close a connection after the 2 FIN packets. This evidence led to a possible conclusion that the flows within the range we selected were those with a unique IP and port combination during the capture. If two or more connections open with the same combination during the capture, they show these unexpected statistics.

Since there is no documentation about the labeling and the authors still need to release the timestamps, we cannot make it suitable for real-world usage. Instead, we decided to show how explainability can be used to expose the simplified test bed.

We also analyzed the Pcap files using the same tools as the CRiSIS 2022 reordercap and editcap with a window of 500 microseconds. We found that 0,03% of the packages were duplicated, and 0,23% were out of order. The authors confirmed that issue and did not perform preprocessing before making the dataset.

* 1. **Model performance**

The following training, classification, and explanation times have been measured on a Dell XPS 13 9315 with a 12th gen i7-1250u and 16gb of ddr5 ram running Fedora 39 with Linux 6.8. For the environment information, there is a YAML file on the GitHub repository with all the versions of the packages in the conda environment. It is crucial to notice that Sklearnex and daal4py have been used to improve inference time, and all the models have been set to use all the cores available.

Table 5.1 - NFS-2023-TE and NFS-2023-nTe parameters

|  |  |  |
| --- | --- | --- |
| mode | parameter | value |
| Random forest | n\_estimators | 10 |
| max\_depth | 14 |
| max\_features | None |
| bootstrap | False |
| catboost | iterations | 40 |
| depth | 11 |
| learning\_rate | 0.4 |
| loss\_function | MultiClass |
| ebm | learning\_rate | 1 |
| lightgbm | objective | MultiClass |
| num\_class | 15 |
| learning\_rate | 0.01 |
| num\_iterations | 250 |
| max\_depth | 4 |
| num\_leaves | 6 |
| xgboost | n\_estimators | 60 |
| max\_depth | 10 |
| objective | multi:softprob |
| learning\_rate | 0.3 |
| Dt | max\_depth | 14 |

The parameter tuning of *Table 5.1* was done following the suggestions of the documentation for each library and by looking at what was working for the other models. We used the metrics f1 macro of sklearn to evaluate the results, and when different parameters were giving similar results, a shorter training time was used to lead the decision.

Table 5.2 - NFS-2023-TE training times



*Table 5.2* shows the training time, explanation time, and the sum of both in ms for each model tested with 6087 samples. Fast tree shap is set to choose the best algorithm for the explanation automatically. These times have some variance but show the order of magnitude necessary to run each model. We can notice that even if EBM is the slowest during training, it can be faster than lightgbm and xgboost if we consider the explanation time. The random forest is close to the decision tree thanks to using only ten trees, which is considerably less than the GAM models. Another thing to consider is that even if Catboost uses fewer trees than the lightgbm and xgboost, it is slower to train.

Table 5.3- NFS-2023-TE – average F1 score of 5-fold



In *Table 5.3*, we compare the prediction time in μs for 1522 samples; in this case, we have used the magic function %timeit of ipython to compute the mean of different runs. Doing the mean is essential because it is statistically more significant. With such a smaller duration, a warmup of the function is necessary to reduce the computation time. Without the warmup run, the classification time of the decision tree was 3 ms because, at the first run, the computer loaded the prediction function into memory, which took more time than the prediction itself.

Looking at the EBM, we notice that it is the best with F1 macro but the worst with F1 weighted. Comparing these results with the decision tree shows that the decision tree has a problem with Hearthbleed and Web Attack - SQL Injection, which misclassifies two samples for the first and 4 with the latter. In comparison, the EBM misclassifies 111 samples in the DoS Slowhttptest. These results expose the need for a balanced dataset; we can argue that EBM is the best one because it can handle each attack well, but on the other hand, building a model over 11 samples will lead to something that will not work in the real world.

Moreover, we need to consider the prediction time; by looking at the F1 results, lightgbm is probably the winner by being the second-best in both macro and weighted scores but the worst in prediction time. Lightgbm is twice as slow as the random forest, which scores nearly the same in the F1 metrics, making the random forest the best alternative overall. The classification time is so important because, in case of a DoS attack that opens and closes the connection at a fast enough speed, the model needs to keep up with each new flow generated; otherwise, the NIDS will run out of service. Another reason why this classification time is necessary is that it can make a difference between enabling or not the use on the edge and lead to less expensive devices when the GPU is not required for running an NIDS.

Table 5.4 - HIKARI 2021 parameters

|  |  |  |
| --- | --- | --- |
| model | parameter | value |
| cat | iterations | 10 |
| depth | 8 |
| learning\_rate | 0.6 |
| loss\_function | MultiClass |
| dt | max\_depth | 8 |
| ebm | default |  |
| lightgbm | objective | MultiClass |
| num\_leaves | 16 |
| n\_estimators | 5 |
| max\_depth | 6 |
| rf | n\_estimators | 5 |
| max\_depth | 10 |
| max\_features | None |
| bootstrap | False |
| xgb | n\_estimators | 3 |
| max\_depth | 8 |
| learning\_rate | 1 |
| objective | multi:softprob |

*Table 5.4* shows how the parameters for HIKARI-2021 lead to less complex models except for the ebm. One of the reasons is that this dataset is balanced after the under-sampling.

Table 5.5- HIKARI-2021 training times



Therefore, *Table 5.5* shows lower training times for the 31952 training samples compared to NFS-2023-TE. EBM is the only model that has increased the training time because, contrary to the other models, it was impossible to build a smaller model with fine-tuning.

Table 5.6 - HIKARI-2021 F1 score and times



*Table 5.6* shows how the model performs nearly the same regarding the F1 score, while the faster model was the decision tree, making it the best model for HIKARI-2021.

* 1. ***Model explanations***

We decided to compare the explanations of the decision tree with the one of the catboost since they decided to give different features importance. Moreover, we printed the decision tree to analyze why tree shap gave more importance to some features.

Figure 5‑1 - NFS-2023-TE - decision tree feature importance with shap of bot class

A graph with red and black text

Description automatically generated

*Figure 5‑1* shows the feature importance of a decision tree for the class bot of NFS-2023-TE. The model has misclassified only 1 sample out of 737 using only a small subset of the available features. The same pattern of using a few features applies to the other attacks. The only exception is benign traffic, which requires more features to be detected. The benign class assigns positive importance to more features because most of the split leads to either attack or benign traffic.

Figure 5‑2 - HIKARI-2021 - decision tree feature importance with shap of bruteforce-XML class

Immagine che contiene testo, schermata, numero, linea

Descrizione generata automaticamente

In *Figure 5‑2*, we can see that the decision tree gives almost all the importance to one feature; with only the first two nodes, the decision tree can classify bruteforce-XML with a gini coefficient of 0.041. This means that 6388 of the 6526 were correctly classified.

Table 5.7 - fwd\_pkts\_payload.max quintiles

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **category** | **min** | **25%** | **50%** | **75%** | **max** |
| **Background** | 0 | 0 | 40 | 127 | 15741 |
| **Benign** | 0 | 0 | 36 | 426 | 3456 |
| **Bruteforce** | 0 | 357 | 373 | 425 | 786 |
| **Bruteforce-XML** | 0 | 746 | 747 | 748 | 925 |
| **Probing** | 517 | 517 | 517 | 517 | 517 |
| **XMRIGCC CryptoMiner** | 40 | 40 | 50 | 50 | 232 |

*Table 5.7* shows the quintile grouped by category for the first feature shown in *Figure 5‑2*, knowing that the tree’s root node selects all the samples less or equal to 737 and that the false branch filters all the samples less or equal to 761.5 made clear why this feature is the most important in detecting a bruteforce-XML attack. The problem is that this quintile distribution of this feature is not valid in the real world. Moreover, changing the payload size can fool a model built with these data.

Figure 5‑3- HIKARI-2021 - decision tree feature importance with shap of XMRIGCC CryptoMiner

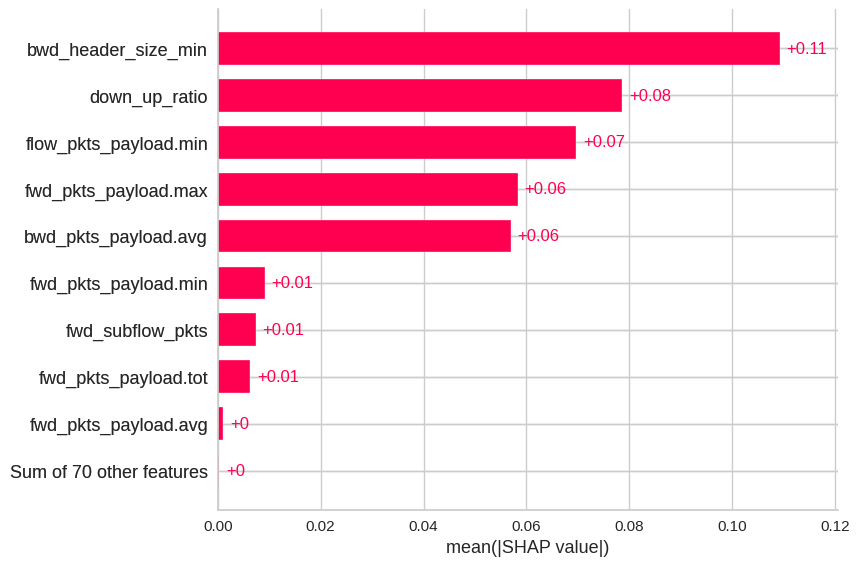


Table 5.8 - Down\_up\_ration mean and std

|  |  |  |
| --- | --- | --- |
| **category** | **mean** | **std** |
| **Background** | 0.810244 | 0.721923 |
| **Benign** | 2.015250 | 32.321667 |
| **Bruteforce** | 8.794107 | 89.440773 |
| **Bruteforce-XML** | 20.071672 | 128.065872 |
| **Probing** | 1.298040 | 0.119266 |
| **XMRIGCC CryptoMiner** | 0.000000 | 0.000000 |

*Figure 5‑3* shows the importance of the features of XMRIGCC CryptoMiner with the decision tree. Again, only a few selected features have been used; the most important is bwd\_header\_size\_min which for this attack is always zero then we have down\_up\_ratio. Table 5.8 shows that the crypto miner is the only one with always 0 for this feature, making it easy to understand why it is essential for this class. Down\_up\_ratio is also the most critical feature for probing, where we can see that the mean value is around 1.3, with a low standard deviation compared to the other attacks. This is the second most important feature for classifying background, but in this case, we do not have any hypothesis as to why.

Figure 5‑4 - NFS-2023-TE - catboost impact on the model output of DDoS with shap sorted by highest magnitude of impact

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

*Figure 5‑4* is the explanation of the catboost algorithm where we can see that the feature with the highest magnitude of all is bidirectional\_fin\_packets, which shows up for Bot, DDoS, DoS Slowhttptest, DoS Slowloris, Heartbleed, Infiltration, and Web Attack – XSS as one of the two most important feature in average or the one with the highest magnitude of impact. By analyzing this feature, we can notice that Bot and Web Attack -XSS always have 1 FIN packet, while DDoS averages 99% of the samples with 1 FIN packet. This characteristic makes the difference in the other attacks cited at the 85th percentile of the distribution showing 0 FIN packets. To understand why we have a lot of 0 we analyzed the duration and for Infiltration and Heartbleed, most of the connection gets closed because of the timeout, while for DoS Slowhttptest and DoS Slowloris, it is hard to tell since they did not reach the timeout either they send an RST packet.

All these analyses conclude that using a more complex model than a decision tree or a random forest is difficult to justify for these datasets. However, this does not mean that the decision tree is ready for deployment in the real world. The reason behind this algorithm's success is the oversimplified test bed.

# **Conclusion**

We first saw that CICIDS-2017 is nowadays one of the most popular datasets cited in Google Scholar, and then we saw how new proposals such as NFS-2023-TE and HIKARI-2021 aim to improve the work done with CIC-IDS 2017. In particular, NFS-2023-TE uses the same raw data as CICIDS-2017, while HIKARI-2021 proposes a new, updated test bed.

Then, we analyzed the flaws in the current literature and the dataset in use, which led to the proposal of a new methodology, with the addition of the explainability part, as seen in other work.

HIKARI-2021 has been analyzed using the criteria proposed by NFS-2023-TE, which led to the discovery of some flaws in the raw data and the dataset generation.

Finally, the analysis of the results shows how comparing simpler models such as the decision tree is necessary to have a benchmark for more complex models. Moreover, we have shown how explainable AI algorithms such as Shap need to be used to help build better models and to increase the accountability of the proposed models. Thanks to the explanations, we discovered something about the dataset, which has helped us avoid mistakes during the work. More than once, features that were not supposed to be in the training set were identified thanks to Shap.

Future improvements must be made to build a model that can be transferred to the real world. Since datasets are far from ready, building complex models using these data does not add any real contribution. This work has added two contributions. We have performed the first analysis of HIKARI-2021, exposing significant issues, and we provided some examples of how tree shap can be used to understand models and datasets better. New studies need to focus on implementing new methodologies, analyzing existing datasets to find areas of improvement, and building these new datasets. Even if the HIKARI-2021 test bed was simplified, making a refinement like NFS-2023-TE for CIC-IDS 2017 is a step in the right direction.

# **Bibliographical References**

Ahmetoglu, S., Che Cob, Z., & Ali, N. (2022). A Systematic Review of Internet of Things Adoption in Organizations: Taxonomy, Benefits, Challenges and Critical Factors. *Appl. Sci. 2022*, *12*, 4117. https://doi.org/10.3390/app12094117

Aouini, Z., & Pekar, A. (2022). NFStream: A flexible network data analysis framework. *Computer Networks*, *204*. https://doi.org/10.1016/j.comnet.2021.108719

Breiman, L. (2001). Random forests. In *Random Forests* (Vol. 45). https://doi.org/10.4324/9781003109396-5

Brownlee, N., Mills, C., & Ruth, G. (1999). *Traffic Flow Measurement: Architecture*.

Catillo, M., Pecchia, A., Rak, M., & Villano, U. (2021). Demystifying the role of public intrusion datasets: A replication study of DoS network traffic data. *Computers & Security*, *108*, 102341. https://doi.org/10.1016/j.cose.2021.102341

Catillo, M., Pecchia, A., & Villano, U. (2023). Machine Learning on Public Intrusion Datasets: Academic Hype or Concrete Advances in NIDS? *Proceedings - 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume, DSN-S 2023*, 132–136. https://doi.org/10.1109/DSN-S58398.2023.00038

Chauhan, R., & Shah Heydari, S. (2020). *Polymorphic Adversarial DDoS attack on IDS using GAN*.

check point. (2021). *CYBER ATTACK TRENDS Mid Year Report 2021*.

Chen, H., Janizek, J. D., Lundberg, S., & Lee, S.-I. (2020). *True to the Model or True to the Data?* https://github.com/slundberg/shap/blob/

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *13-17-August-2016*, 785–794. https://doi.org/10.1145/2939672.2939785

Dorogush, A. V., Ershov, V., & Yandex, A. G. (2017). *CatBoost: gradient boosting with categorical features support*. https://github.com/Microsoft/LightGBM

Engelen, G., Rimmer, V., & Joosen, W. (2021). Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. *Proceedings - 2021 IEEE Symposium on Security and Privacy Workshops, SPW 2021*, 7–12. https://doi.org/10.1109/SPW53761.2021.00009

Fernandes, R., & Lopes, N. (2022). Network Intrusion Detection Packet Classification with the HIKARI-2021 Dataset: a study on ML Algorithms. *10th International Symposium on Digital Forensics and Security, ISDFS 2022*, 1–5. https://doi.org/10.1109/ISDFS55398.2022.9800807

Fernandes, R., Silva, J., Ribeiro, O., Portela, I., & Lopes, N. (2023). The impact of identifiable features in ML Classification algorithms with the HIKARI-2021 Dataset. *ISDFS 2023 - 11th International Symposium on Digital Forensics and Security*, 1–5. https://doi.org/10.1109/ISDFS58141.2023.10131864

Ferriyan, A., Thamrin, A. H., Takeda, K., & Murai, J. (2021). Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic. *Applied Sciences (Switzerland)*, *11*(17). https://doi.org/10.3390/app11177868

Gharib, A., Sharafaldin, I., Habibi, A., & Ghorbani, A. (2016). *An Evaluation Framework for Intrusion Detection Dataset*.

Goodman, B., & Flaxman, S. (2017). *European Union Regulations on Algorithmic Decision Making and a “Right to Explanation.”*

Guerra, J. L., Catania, C., & Veas, E. (2022). Datasets are not enough: Challenges in labeling network traffic. *Computers & Security*, *120*, 102810. https://doi.org/10.1016/j.cose.2022.102810

Gümüşbaş, D., Yıldırım, T., Genovese, A., & Scotti, F. (2021). A comprehensive survey of databases and deep learning methods for cybersecurity and intrusion detection systems. In *IEEE Systems Journal* (Vol. 15, Issue 2, pp. 1717–1731). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/JSYST.2020.2992966

Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX. *IEEE Communications Surveys and Tutorials*, *16*(4), 2037–2064. https://doi.org/10.1109/COMST.2014.2321898

Janzing, D. (2019). *Causal Regularization*.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *31st Conference on Neural Information Processing Systems*. https://github.com/Microsoft/LightGBM.

Kurniabudi, Stiawan, D., Darmawijoyo, Bin Idris, M. Y. Bin, Bamhdi, A. M., & Budiarto, R. (2020). CICIDS-2017 Dataset Feature Analysis with Information Gain for Anomaly Detection. *IEEE Access*, *8*, 132911–132921. https://doi.org/10.1109/ACCESS.2020.3009843

Kwon, D., Neagu, R. M., Rasakonda, P., Ryu, J. T., & Kim, J. (2023). Evaluating Unbalanced Network Data for Attack Detection. *SNTA 2023 - Proceedings of the 2023 on Systems and Network Telemetry and Analytics*, 23–26. https://doi.org/10.1145/3589012.3594898

Lanvin, M., Gimenez, P. F., Han, Y., Majorczyk, F., Mé, L., & Totel, É. (2023). Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13857 LNCS*, 18–33. https://doi.org/10.1007/978-3-031-31108-6\_2

Lashkari, A. H., Gil, G. D., Mamun, M. S. I., & Ghorbani, A. A. (2017). Characterization of tor traffic using time based features. *ICISSP 2017 - Proceedings of the 3rd International Conference on Information Systems Security and Privacy*, *2017-Janua*, 253–262. https://doi.org/10.5220/0006105602530262

Lee, W., Stolfo, S. J., & Mok, K. W. (1999). *Mining in a Data-flow Environment: Experience in Network Intrusion Detection*.

Liu, J., Gao, Y., & Hu, F. (2021). A fast network intrusion detection system using adaptive synthetic oversampling and LightGBM. *Computers and Security*, *106*, 102289. https://doi.org/10.1016/j.cose.2021.102289

Liu, L., Engelen, G., Lynar, T., Essam, D., & Joosen, W. (2022). Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. *2022 IEEE Conference on Communications and Network Security, CNS 2022*, 254–263. https://doi.org/10.1109/CNS56114.2022.9947235

Lou, Y., Caruana, R., Gehrke, J., & Hooker, G. (2013). Accurate intelligible models with pairwise interactions. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Part F1288*, 623–631. https://doi.org/10.1145/2487575.2487579

Louk, M. H. L., & Tama, B. A. (2023). Dual-IDS: A bagging-based gradient boosting decision tree model for network anomaly intrusion detection system. *Expert Systems with Applications*, *213*, 119030. https://doi.org/10.1016/j.eswa.2022.119030

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., & Lee, S. I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*(1), 56–67. https://doi.org/10.1038/s42256-019-0138-9

Lundberg, S. M., Erion, G. G., & Lee, S.-I. (2018). *Consistent Individualized Feature Attribution for Tree Ensembles*. http://github.com/slundberg/shap

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, *2017-Decem*, 4766–4775. https://github.com/slundberg/shap

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021). Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset. *IEEE Access*, *9*, 22351–22370. https://doi.org/10.1109/ACCESS.2021.3056614

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, 1–6. https://doi.org/10.1109/MilCIS.2015.7348942

Nazar, M., Alam, M. M., Yafi, E., & Su’Ud, M. M. (2021). A Systematic Review of Human-Computer Interaction and Explainable Artificial Intelligence in Healthcare with Artificial Intelligence Techniques. In *IEEE Access* (Vol. 9, pp. 153316–153348). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2021.3127881

Noori, M. S., Sahbudin, R. K. Z., Sali, A., & Hashim, F. (2023). Feature Drift Aware for Intrusion Detection System using Developed Variable Length Particle Swarm Optimization in Data Stream. *IEEE Access*, *11*(September), 1–1. https://doi.org/10.1109/access.2023.3333000

Panigrahi, R., & Borah, S. (2018). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering and Technology(UAE)*, *7*(3.24 Special Issue 24), 479–482. https://www.researchgate.net/publication/329045441

Pekar, A., & Jozsa, R. (2024). *Evaluating ML-Based Anomaly Detection Across Datasets of Varied Integrity: A Case Study*. https://intrusion-detection.distrinet-research.be/WTMC2021/tools

Rajak, P., Lachure, J., & Doriya, R. (2022). CNN-LSTM-based IDS on Precision Farming for IIoT data. *Proceedings of 4th International Conference on Cybernetics, Cognition and Machine Learning Applications, ICCCMLA 2022*, 99–103. https://doi.org/10.1109/ICCCMLA56841.2022.9988997

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *“Why Should I Trust You?” Explaining the Predictions of Any Classifier*. https://doi.org/10.1145/2939672.2939778

Sahakyan, M., Aung, Z., & Rahwan, T. (2021). Explainable Artificial Intelligence for Tabular Data: A Survey. *IEEE Access*, *9*, 135392–135422. https://doi.org/10.1109/ACCESS.2021.3116481

Sarhan, M., Layeghy, S., & Portmann, M. (2022). Evaluating Standard Feature Sets Towards Increased Generalisability and Explainability of ML-Based Network Intrusion Detection. *Big Data Research*, *30*, 100359. https://doi.org/10.1016/j.bdr.2022.100359

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, *2018-Janua*(Cic), 108–116. https://doi.org/10.5220/0006639801080116

Shrikumar, A., Greenside, P., Shcherbina, A. Y., & Kundaje, A. (2017). *Not Just A Black Box: Learning Important Features Through Propagating Activation Differences*. https://arxiv.org/

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, *Cisda*, 1–6. https://doi.org/10.1109/CISDA.2009.5356528

TeleGeography. (2023). *Executive Summary, IP Networks Research Service*.

Walter, L., Denter, N. M., & Kebel, J. (2022). A review on digitalization trends in patent information databases and interrogation tools. *World Patent Information*, *69*, 102107. https://doi.org/10.1016/j.wpi.2022.102107

Xiong, F., Zang, L., & Gao, Y. (2021). *Internet penetration as national innovation capacity: worldwide evidence on the impact of ICTs on innovation development*. https://doi.org/10.1080/02681102.2021.1891853

Yang, J. (2021). *Fast TreeSHAP: Accelerating SHAP Value Computation for Trees*. http://arxiv.org/abs/2109.09847

Yulianto, A., Sukarno, P., & Suwastika, N. A. (2019). Improving AdaBoost-based Intrusion Detection System (IDS) Performance on CIC IDS 2017 Dataset. *Journal of Physics: Conference Series*, *1192*(1). https://doi.org/10.1088/1742-6596/1192/1/012018

Zavrak, S., & Iskefiyeli, M. (2020). Anomaly-Based Intrusion Detection from Network Flow Features Using Variational Autoencoder. *IEEE Access*, *8*, 108346–108358. https://doi.org/10.1109/ACCESS.2020.3001350

Zhang, Z., Hamadi, H. Al, Damiani, E., Yeun, C. Y., & Taher, F. (2022). Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research. *IEEE Access*, *10*(July), 93104–93139. https://doi.org/10.1109/ACCESS.2022.3204051

# **Appendix A**

Source code available at: <https://github.com/ludotosk/tesi>

A picture containing text, screenshot, font, design

Description automatically generated