TECHNISCHE UNIVERSITÄT DARMSTADT

MACHINE LEARNING MODELS *in* NETWORK INTRUSION DETECTION SYSTEMS

Self-Supervised Detection of Malicious Flows and Traffic Patterns Recognition in Programmable Networks

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

PEGAH GOLCHIN, M.SC.

Vorsitz: Prof. Dr. rer. nat. Sascha Preu
Referent: Prof. Dr.-Ing. Dr. h.c. Ralf Steinmetz
Korreferent: Prof. Dr. Andreas Mauthe

Tag der Einreichung: 7. Mai 2024
Tag der Disputation: 16. Juli 2024

Darmstadt 2024

Pegah Golchin, M.Sc.: *Machine Learning Models in Network Intrusion Detection Systems,*
Self-Supervised Detection of Malicious Flows and Traffic Patterns Recognition
in Programmable Networks

Darmstadt, Technische Universität Darmstadt

Jahr der Veröffentlichung der Dissertation auf TUprints: 2024
Tag der mündlichen Prüfung: 16. Juli 2024

ABSTRACT

The recent increase in cyber-attacks highlights the critical need for reliable Network Intrusion Detection Systems capable of detecting anomalies before they inflict substantial damage. Conventional intrusion detection methods often fail to classify previously unseen intrusion patterns accurately. This shortfall is exacerbated by the emergence of new network intrusion types and the evolving nature of network structures. Machine Learning (ML) models address this need by learning representations of network traffic flows. Nonetheless, challenges persist, particularly in ensuring their adaptability and ability to generalize in detecting various network traffic patterns and integrating them into programmable networks.

The first contribution of this thesis highlights the presence of diverse flow feature patterns in existing network traffic patterns. To mitigate the impact of these disparities on the final detection performance and minimize noise in flow features, thereby reducing the complexity of ML models, an Ensemble Feature Selection approach is devised. This method integrates statistical and ML-based feature selectors, taking into account the imbalance of benign and attack traffic to avoid biased feature extraction. Evaluation results demonstrate the potential to attain high detection performance with a reduced flow feature dimension. Additionally, a data-driven approach is incorporated into the proposed feature selection method to improve the transferability of selected flow features across different network traffic patterns.

The second contribution aimed at tackling two main challenges: the limited availability of annotated network traffic flow data required for training ML models and the limited ability of ML models to generalize across various network traffic patterns. To overcome these challenges, a Self-Supervised Contrastive Learning approach is introduced, which is specifically trained on benign flows to learn the abstract representation of benign flow patterns. The results illustrate improvements in the generalization of detection performance across diverse network traffic patterns. These improvements surpass the performance of both supervised and unsupervised ML models used as baselines.

The last contribution explores integrating ML models into programmable networks, particularly following the Software-Defined Networking paradigm, which separates the data plane from the control plane. However, deploying complex ML models in the control plane can increase the risk of overwhelming it, given the necessity to forward flows through it. Conversely, employing lightweight models with few trainable parameters in the data plane may compromise detection performance. To tackle these challenges, we propose a collaborative ML-based intrusion detection approach. This approach facilitates cooperation between ML models deployed in the data plane and the control plane based on the confidence level of the deployed ML model in the data plane. Using this approach, a balance is achieved between attaining high detection performance and speed while reducing network load.

KURZFASSUNG

Die jüngste Zunahme von Cyberangriffen verdeutlicht den zwingenden Bedarf von zuverlässigen Systemen zur Erkennung von Eindringlingen in Netzwerken. Solche Systeme müssen diese Eindringlinge erkennen, bevor deren Angriffe erheblichen Schaden anrichten können. Herkömmliche Methoden zur Erkennung von Eindringlingen sind oft nicht in der Lage, neue oder bisher unbekannte Angriffsmuster genau zu klassifizieren. Dieses Manko wird durch das vermehrte Auftreten neuartiger Angriffsversuche und die sich verändernden Netzwerkstrukturen noch verschärft. Methoden des maschinellen Lernens (ML) addressieren dieses Problem, indem sie charakteristische Eigenschaften von Netzwerkverkehr lernen. Dennoch bestehen weitere Herausforderungen, insbesondere bei der Gewährleistung ihrer Anpassungsfähigkeit und Generalisierbarkeit bei der Erkennung verschiedener Netzverkehrsmuster und ihrer Integration in programmierbare Netzwerke.

Der erste Beitrag dieser Arbeit behandelt die Existenz verschiedener Muster von Verkehrsflussmerkmalen. Um die Auswirkungen dieser Unterschiede auf die endgültige Erkennungsleistung abzuschwächen, das Rauschen in den Verkehrsflussmerkmalen zu minimieren und damit die Komplexität der ML-Modelle zu reduzieren, ist ein Ensemble-Feature-Selection-Ansatz entwickelt worden. Diese Methode integriert statistische und ML-basierte Merkmalsselektoren und berücksichtigt das Ungleichgewicht zwischen gutartigen und bösartigen Verkehrsdaten, um eine verzerrte Merkmalsextraktion zu vermeiden. Die Evaluationsergebnisse zeigen, dass mit einer reduzierten Merkmalsdimension dennoch eine sehr hohe Erkennungsleistung unter Berücksichtigung aller Flussmerkmale erreichbar ist. Um die Übertragbarkeit ausgewählter Verkehrsflussmerkmale auf verschiedene Verkehrsmuster im Netz zu verbessern, wird ein datengesteuerter Ansatz in die vorgeschlagene Methode zur Merkmalsauswahl integriert.

Der zweite Beitrag umfasst weitere Untersuchungen zur Bewältigung zweier Herausforderungen: die begrenzte Verfügbarkeit von annotierten Netzwerkverkehrsflussdaten, die für das Training von ML-Modellen erforderlich sind, und die Fähigkeit von ML-Modellen, über verschiedene Netzwerkverkehrsmuster hinweg zu generalisieren. Um diese Herausforderungen zu bewältigen, wird Contrastive Self-Supervised Learning genutzt, das speziell auf gutartigen Datenflüssen trainiert wird, um die abstrakte Darstellung gutartiger Datenflussmuster zu erlernen. Die Ergebnisse zeigen Verbesserungen bei der Generalisierung der Erkennungsleistung über verschiedene Netzwerkverkehrsmuster hinweg. Diese Verbesserungen übertreffen die Leistung sowohl überwachter als auch unüberwachter ML-Modelle, die als Vergleichsansätze verwendet werden.

Der letzte Beitrag befasst sich mit der Integration von ML-Modellen in programmierbaren Netzen, insbesondere in Anlehnung an das Paradigma des Software-Defined Networking, in welchem die Datenebene von der Kontrollebene getrennt ist. Der Einsatz komplexer ML-Modelle in der Kontrollebene kann diese jedoch überlasten, da Datenströme durch das Kontrollnetzwerk weitergeleitet werden müssen. Dagegen vermindert die Verwendung leichtgewichtiger Modelle mit wenigen trainierbaren Parametern in der Datenebene die Erkennungsleistung negativ. Um diese Herausforderungen zu bewältigen, wird in dieser Arbeit ein kollaborativer ML-basierter Ansatz zur Erkennung von Eindringlingen vorgeschlagen. Dieser Ansatz ermöglicht die Zusammenarbeit von ML-Modellen auf der Datenebene und der Kontrollebene, basierend auf dem Vertrauensniveau des ML-Modells in der Datenebene. Mit diesem Ansatz wird eine Balance zwischen hoher Erkennungsleistung und Geschwindigkeit bei gleichzeitiger Verringerung der Netzwerklast des Kontrollnetzwerks erreicht.

PREVIOUSLY PUBLISHED MATERIAL

This thesis incorporates content that has been previously published in scientific conferences. Table 1 outlines the relevant previous publications, none of which has been directly reprinted in this thesis, except for tables and figures, specifically for evaluation sections. Each source is clearly identified by providing its reference in the corresponding caption. A comprehensive list of the author's publications is available in Appendix C.

Throughout this dissertation, I have acknowledged science as a collaborative effort where all the findings are derived from teamwork. Therefore, I would like to extend my gratitude and acknowledge the contributions of all the relevant collaborators, co-authors, and their respective affiliations. In cases where no specific affiliation is mentioned, it should be noted that the individual is or has been a colleague at the Multimedia Communications Lab of the Technical University of Darmstadt. In the following chapters, the pronoun "*we*" will be used to acknowledge the collaborative team effort.

Table 1: Previously publications in each thesis chapter

Chapter	Publications
1	[57], [61], [60], [58]
2	[57], [61], [60], [58], [56]
3	[57],[59],[60]
4	[58]
5	[61]

Chapter 2 offers a comprehensive overview of the related work in Machine learning-based Network Intrusion Detection Systems (ML-based NIDS) and examines the state-of-the-art to identify pertinent research gaps. During the preparation of previous publications, I conducted several literature reviews for the respective related work sections. Consequently, a significant portion of the related work analysis has already been published, with the primary analysis outlined in [57], [58], and [61]. The related work chapter of the thesis is a refined, reorganized, and restructured amalgamation of these sections. To address the research gaps, I received valuable support from Prof. Dr.-Ing. Dr. h.c. Ralf Steinmetz, Dr.-Ing. Ralf Kundel, Dr.-Ing. Tobias Meuser and Dr. Nima Rafiee (Zalando, Berlin) through discussions on relevant papers and regular meetings. Their insights contributed to the depth and clarity of the analysis.

Chapter 3 conducts a comprehensive exploratory data analysis of several public network traffic datasets, pinpointing research gaps concerning feature selection methods employed in ML-based NIDS. To shed light on these gaps, I collaborated with Nima Rafiee, Dr.-Ing. Tim Steuer, and Jannis Weil, M.Sc., to explore artificial intelligence aspects and with Ralf Kundel to examine network communication perspectives. Our collaborative efforts culminated in [57], [60], and [59] publications, which further elucidated our findings. Ralf Steinmetz supervised this research project and scientific positioning. To ensure the quality of our scientific publications relevant to this chapter, all co-authors thoroughly review the papers and provide valuable feedback. This collaborative effort helps refine and improve the publications, incorporating diverse perspectives and expertise. The Ensemble Feature Selection (EFS) method, introduced in paper [57], serves as the foundation for the evaluations conducted in Chapter 3 (Section 3.7) of this dissertation. However, the evaluations presented in this chapter extend beyond those in [57], encompassing evaluations on a broader range of network traffic datasets. To improve the transferability of the feature selection approach, a data-driven solution is integrated into the EFS method, termed DD-EFS [59]. The feature importance illustrated in *Figure 8* and evaluation results demonstrated in *Table 6* are reprinted from [59].

Chapter 4 addresses the challenges posed by the imbalance and scarcity of annotated malicious flow data in network traffic datasets, aiming to enhance the generalization of ML models' detection performance. A Self-Supervised Contrastive Learning approach is devised, and its efficacy across multiple network traffic datasets is investigated. The conceptualization of this approach involved regular meetings with Nima Rafiee. The findings of this research are detailed in [58], a publication that underwent thorough review and feedback from all co-authors, Ahmad Khalil, M.Sc., Mehrdad Hajizadeh, M.Sc. (Communication Networks, Technical University of Chemnitz), and Ralf Kundel to enhance its quality. Ralf Kundel and Ralf Steinmetz supported and advised me. The method presented in this chapter is fully described in [58], and the *algorithm 2* is the same as the one available in the paper [58]. Additionally, *Section 4.6.2, 4.8.2, 4.8.3, and 4.8.7* are taken almost verbatim from [58]. Furthermore, the selected hyperparameters in *Section 4.8.1* and evaluation metrics in *Section 4.7.4*, and the figures and tables in the evaluation section of this chapter (*Section 4.8*) are identical to those in [58]. Nevertheless, in this dissertation, each figure and table is elucidated with more detailed explanations to provide a deeper understanding of the evaluation process and results.

The concept of deploying a lightweight ML model with few trainable parameters in programmable networks, as discussed in **Chapter 5**, originated from a master thesis I proposed to Yizi Liu, M.Sc.. Through analyzing the results and related literature, I hypothesized that relying solely on an ML model in the programmable data plane might not achieve optimal detection performance while deploying it solely in the control plane could increase network load and latency. These insights prompted me to propose a collaborative approach between them, leveraging the confidence of the deployed ML model in the programmable data plane. I proposed this idea for a master thesis to Chengbo Zhou, M.Sc., who completed his master thesis under my supervision. His thesis was awarded the Best Master Thesis at the Multimedia Communications Lab. The outcomes of this thesis were subsequently expanded and published in [61], receiving the best paper award. The conception of this work originated completely from my ideas, and I developed the ML models and preprocessing pipelines used. Chengbo Zhou also actively made contributions to the implementation of this work. The paper was written mainly by me. Chengbo Zhou, Pratyush Agnihotri, M.Sc., Mehrdad Hajizadeh, and Ralf Kundel provided feedback on the manuscript. Ralf Steinmetz supervised me scientifically during this work. The collaborative machine learning-based intrusion detection system (CML-IDS) concept is also available in [61], and the general architecture of CML-IDS depicted in *Figure 18* can be found in [61] as well. Similarly, the figures and tables showcased in the evaluation section (*Section 5.8*) of this dissertation are reproduced from [61]. Additionally, *Figure 21* and *Figure 24* are reproduced from Chengbo Zhou's master thesis. Moreover, *Section 5.8.9* is taken almost verbatim from [61].

The aforementioned publications underwent thorough peer-review processes involving numerous anonymous reviewers who provided invaluable feedback. I incorporated their suggestions into the final versions of the papers and articles, thus indirectly contributing to this thesis. I would like to express my gratitude once again to all reviewers for their invaluable contributions.

This thesis reflects my independent work, and certain sections have been linguistically reviewed using the assistance of tools like Grammarly and ChatGPT. These tools were utilized specifically to refine the grammatical structure and enhance the clarity of expression.

CONTENTS

1 INTRODUCTION

The pervasive growth of Internet connectivity, with nearly two-thirds of the world's population projected to reach by 2023, underscores the deep integration of the Internet into our daily lives [34]. These interconnections have led to unparalleled technological progress, driving economic growth and innovation. However, the digital revolution has paved the way for increased cyber security risks [81]. According to Cisco's annual report, the number of Distributed Denial of Service (DDoS) attacks with a peak attack traffic between 100 Gbps and 400 Gbps increased by 776% from 2018 to 2019 [34]. Additionally, findings from a recent study [168] highlight a growing trend in the frequency of cyber attacks, revealing a doubling in the total number of DDoS attacks from 7.9 million in 2018 to 15.4 million by 2023.

If such cyber attacks target a company, it can result in significant financial losses and damage to its reputation [5]. For instance, Amazon Web Services (AWS) experienced a massive DDoS attack that reached a peak traffic of 2.3 terabits per second in 2020 [14]. This attack temporarily disrupted the services provided by AWS Shield. Although AWS successfully mitigated the attack, this incident highlighted the critical role of detecting intrusions in preventing substantial financial losses and service disruptions [6]. Therefore, there is a need for Network Intrusion Detection Systems (NIDSs) that can detect such network anomalies rapidly.

A NIDS resides within the network and actively monitors network traffic for signals of suspicious activities [53]. Various methodologies can be employed in the development of a NIDS to enhance its efficacy in identifying potential threats [165]. In a NIDS, one important metric is its detection performance. This refers to how accurately the system can classify network traffic as either benign or attack (i.e., malicious) flows.

Traditional detection techniques within a NIDS often rely on exact pattern matching to identify malicious activities, drawing comparisons with previously recorded attacks [1]. However, with the continuous evolution of cyber threats and the emergence of novel malicious patterns, there is a pressing need for the NIDS to detect not only known attacks but also new and previously unseen malicious patterns. This can enhance their capability to detect a broader of cyber attacks [117].

Machine Learning (ML) models can meet the requirement of classifying unseen traffic flows by leveraging statistical features extracted from flow data [42, 174]. These models possess the ability to discern intricate patterns and anomalies within network traffic, thereby empowering them to effectively detect potential threats, even those that have not been previously encountered [53]. Moreover, as cyber threats are not confined to traditional distributed network architectures and can also occur within emerging network programming paradigms like Software-Defined Networking (SDN), there is a critical need for adaptable NIDSs. ML-based NIDSs fulfill this requirement by continuously learning from new data, enabling them to recognize evolving patterns and anomalies in network traffic within SDN environments.

1.1 MOTIVATION FOR USING MACHINE LEARNING IN MALICIOUS PATTERN DETECTION

Anomaly-based NIDSs aim to define the normal behavior of network traffic and protect the network whenever the deviation from the expected normal behavior exceeds a predefined threshold [53]. Anomaly detection techniques are widely used in NIDSs and can identify both known and unknown attacks, thereby proving to be more effective than signature-based techniques. Additionally, these techniques are useful in creating new signatures for signature-based NIDSs [131].

Statistical-based techniques, such as univariate, multivariate, and time series models, involve analyzing network traffic to create a profile, including its stochastic behavior [15, 68]. These methods can learn the expected behavior of the system through observations. Furthermore, they can provide notifications of malicious activities occurring over extended periods. However, configuring the values of various parameters poses a challenge, particularly due to the delicate balance required between false positives and false negatives [15, 55]. Furthermore, many of these statistical methods rely on the assumption of a quasi-stationary process, which may not always align with real-world scenarios and emerging unknown attack types [25].

ML models can overcome the limitations of statistical models by leveraging historical traffic patterns to learn the statistical features inherent in network traffic patterns. This enables them to effectively discern the relationship between features and attack or benign labels [132]. Furthermore, with the emergence of various new attack behaviors, ML models demonstrate better detection performance compared to statistical approaches like univariate and multivariate models [25].

Cyber threats are not limited to traditional networks but also pose risks to programmable network architectures like SDN [114]. SDN separates the control plane and data plane, enhancing network management flexibility [83]. However, while the logically centralized structure of SDN improves network management, it can be vulnerable to cyber attacks [100]. Consequently, attacks on the control plane can inflict significant damage on the entire SDN infrastructure. Thus, rapid and effective detection in SDN is crucial for preventing potential damage. ML-based NIDSs can fulfill these requirements and detect malicious patterns in SDN as well, especially in the SDN control plane, where abundant computational resources are accessible.

While ML-based NIDSs can detect malicious patterns and are adaptable for deployment in programmable networks, their detection performance is closely tied to their training strategy, model complexity, and the network traffic datasets used during training. Furthermore, the complexity of evolving cyber threats, particularly their ability to closely mimic benign flows, as evidenced in Multi-Stage Attacks and dynamically adaptive botnets, presents additional challenges for precisely detecting malicious patterns [12, 99]. Additionally, deploying ML-based NIDS in SDN requires careful consideration of the classifier placement. This is crucial as improper placement can potentially overwhelm the control plane and decrease detection performance and speed.

Given these issues, evaluating the generalization ability of ML models becomes crucial. In machine learning, generalization involves assessing a model on new and previously unseen data, which may exhibit distribution variations from the training data [43]. This assessment can demonstrate the effectiveness of an ML-based NIDS in classifying previously unseen network flows, regardless of their similarity to their training data distribution. Moreover, exploring strategies for deploying an ML-based NIDS in SDN that balances detection performance, detection speed, and network load opens a relevant research field to fulfill the demands of next-generation NIDSs.

1.2 RESEARCH CHALLENGES

Even though ML-based NIDSs can meet the basic requirements of detecting new attacks by learning the network traffic pattern, several challenges regarding their model complexity level, detection performance generalization, and deployment in programmable networks like SDN exist. The challenges associated with ML-based NIDSs, which will be the primary focus of this thesis, will be briefly explained.

Challenge: *Extracting relevant features from imbalanced network traffic datasets with varying flow patterns*

Various studies demonstrate that supervised ML models perform better in detecting attacks when trained on less noisy features [2, 87, 136, 182]. In the field of network traffic, different types of flow features fall into categories such as time-related, packet-related, protocol-specific, directional-based, and statistical features. Each category could be important for a specific attack, depending on the attack's purpose, target, and damage type [69]. Additionally, network traffic datasets frequently exhibit imbalances, comprising a disproportionate distribution of benign and attack flows. This imbalance often results in a scarcity of data for specific classes within the dataset. The imbalance between attacks and benign flows, along with the intrinsic differences among flow patterns, makes it difficult to extract the most relevant feature set. Hence, addressing these challenges and selecting relevant features that can be applied to various network patterns while reducing the ML model's complexity is crucial.

Challenge: *Generalizing detection performance of ML-based NIDSs across diverse network traffic patterns*

Supervised ML models exhibit high detection performance when encountering previously unseen data from the same distribution as their training dataset [3, 39, 89, 162]. However, they rely on annotated data for training, which restricts their ability to detect flows that diverge significantly from the patterns in their training datasets. Due to the growing number of network applications and emerging technologies, the nature of network traffic has become more dynamic than ever before. Capturing and annotating such dynamic traffic has become a challenging task, if not impossible

[67]. This poses a challenge to the efficacy of supervised ML-based NIDS in detecting new and unseen traffic flows with different distributions. Moreover, advanced attack types such as SlowDoS and Botnets exhibit behavior that is very similar to benign flows making it difficult for ML models to detect them without learning the abstract representation of benign flows [12, 99]. All these challenges indicate a high likelihood of receiving new flows or attacks that have a different distribution from the training dataset of a supervised ML-based NIDS, which can reduce its detection performance. This challenge is called cross-domain detection performance [39, 94], which requires an innovative approach to increase the generalization of the ML-based NIDS for detecting different attack types with varying distributions across different network traffic patterns.

Challenge: *Balancing network Load, detection Performance, and speed in the ML-Based NIDS deployment within SDN*

The usage of the ML-based NIDS extends beyond traditional networks to encompass SDN, where potential intrusion can target the SDN control plane, data plane switches, and users [35, 41, 83, 147]. ML models demand computational resources, making the control plane a logical embedding location. However, this choice necessitates forwarding all flows to the control plane for classification, increasing the risk of overwhelming it and directing attack traffic towards it. The emergence of programmable switches, such as P4 switches, offers a solution by enabling the deployment of lightweight ML models aligned with the match-action tables of these switches [23, 98]. While this approach addresses the challenges associated with embedding ML-based NIDS in the control plane and enhances detection speed to line rate, the detection performance of the ML-based NIDS can be decreased because of the constrained capabilities of lightweight ML models. Therefore, striking a balance between the detection performance, network load, and detection speed of the deployed ML-based NIDS within SDN is needed.

1.3 RESEARCH GOALS AND CONTRIBUTIONS

The main goal of this thesis is to investigate how to achieve high detection performance on intra-dataset and inter-dataset flows utilizing ML-based NIDS and effectively integrate it within programmable networks. To accomplish these, we specify the following research goals.

Research Goal 1: *Selecting the most relevant flow features*

Eliminating non-relevant flow features can reduce the complexity of ML-based NIDS while retaining detection performance. An appropriate feature selection method extracts relevant flow features for diverse network traffic types. Training ML models on this precise feature set enhances the model's generalization ability while reducing training time. In this thesis, we develop an ensemble feature selection method to identify the most relevant features for effectively classifying between attacks and benign flows [57]. The proposed ensemble feature selection approach takes into account the dataset's class imbalance to prevent bias towards the majority class. Furthermore, a data-driven approach is integrated into the proposed ensemble feature selection approach to enhance the transferability of the selected features.

Research Goal 2: *Increasing the cross-domain detection performance of ML-based NIDSs*
The accurate detection of previously unseen flow patterns is a critical evaluation metric for an ML-based NIDS. In our research goal, we aim to enhance the detection performance and generalization of the ML-based NIDS across network traffic flows with distributions different from those in the training dataset. To achieve this, we need an ML model capable of learning abstract representations of benign flows to effectively identify attacks that exhibit distinct behavior from benign flows. Additionally, we aim to decrease the reliance of ML models on labeled data, given the scarcity of annotated datasets [58].

Research Goal 3: *Increasing detection performance of an ML-based NIDS in SDN, while decreasing network load*
The separation of the control plane and data plane in SDN architecture can make the network vulnerable to security threats. However, deploying an ML-based NIDS solely in the control plane can lead to an increase in network load as it entails forwarding flows to the control plane for classification tasks. On the other hand, deploying ML-based NIDS exclusively in the data plane reduces detection performance due to the use of a lightweight ML model. In this research goal, we aim to develop a framework that reduces the number of forwarded flows to the control plane, which in turn reduces the risk of attacks against the control plane and minimizes network overload [61]. We also aim to increase detection speed compared to solely deploying the ML model in the control plane. Furthermore, we aim to achieve higher detection performance within this framework, which is crucial for an ML-based NIDS.

In the following, we briefly discuss topics that fall outside the scope of this thesis. In this thesis, our focus is on the domain of offline ML-based NIDS, where models are trained once and deployed for subsequent detection tasks. However, the field of online learning, where the model updates its training, presents its own set of challenges related to concept drifts, adaptability, and the method of integration of updated information [60].

A Hybrid NIDS, combining both signature-based and anomaly-based approaches, can leverage the strengths of both methodologies. Their synergy can improve the robustness of the detection performance and utilize their complementary capabilities [133]. Nonetheless, this thesis primarily concentrates on exploring key metrics within the realm of ML-based NIDS.

Furthermore, in our contribution to designing a collaborative ML-based NIDS in SDN, we deploy the data plane's ML-based NIDS in BMv2, a softwarized P4 switch. However, deploying an ML model on a physical P4 switch, such as Tofino, can pose new challenges due to the limited number of registers available on such switches. Although we have considered the hardware constraints and have limited the number of registers in our framework, it can still impact the scalability and effectiveness of the ML-based NIDS in Tofino switches.

1.4 STRUCTURE OF THE THESIS

After a brief introduction to this thesis, we focus on describing the necessary background and previous works regarding ML-based NIDS in traditional networks and SDN paradigms in Chapter 2. In Chapter 3, we demonstrate various network traffic data analyses to highlight the differences between network traffic patterns. Based on the findings, we introduce an ensemble feature selection approach aimed at reducing flow feature dimensions and selecting a transferable flow feature set. Chapter 4 demonstrates the limitations of supervised ML-based NIDS in detecting out-of-distribution flow patterns. In this chapter, we introduce a new self-supervised contrastive learning approach that exclusively trains on benign flows and enhances the generalization of detection performance. In Chapter 5, we explain the network security challenges in SDN and propose our collaborative framework that can take advantage of collaborating between deployed ML-based NIDS in both data and control planes. The thesis concludes in Chapter 6 with a brief summary of the core contributions. Lastly, we provide an outlook on potential future work.

2 BACKGROUND AND RELATED WORK

This chapter offers a concise exploration of various Machine Learning (ML) categories applicable to Network Intrusion Detection Systems (NIDS). It also delves into the Software-Defined Networking paradigm and the programmable data plane. After introducing the thesis background, the chapter explores related work relevant to each contribution. Its objective is to provide both a comprehensive overview of current advancements and to highlight gaps in solutions for identified issues.

2.1 INTEGRATING MACHINE LEARNING INTO INTRUSION DETECTION

ML models have shown their efficacy in fulfilling the demands of a NIDS by discerning anomalies in network behavior or classifying network traffic. These models learn patterns within network flows, enabling them to detect both known and previously unseen network intrusions based on deviations from normal behavior [53, 134].

Integrating ML approaches into NIDSs necessitates deploying sequential modules as a pipeline to detect intrusions within network traffic. Figure 1 illustrates the general architecture of an ML-based NIDS, comprising several interconnected modules, each playing a crucial role in the overall detection process.

The input to an ML-based NIDS comprises network traffic data in a Packet Capture (PCAP) format collected from various sources such as network sensors or log files. In this thesis, we used publicly available network traffic datasets. The information about them is available in Appendix A.2. The raw input is converted to flow feature files, which is understandable for ML models. The feature files undergo a preprocessing pipeline where it is cleaned, transformed, and normalized to prepare features for further analysis. Subsequently, feature selection techniques are applied to identify the most relevant and informative attributes from the preprocessed data, reducing dimensionality and focusing the model's attention on discriminative features. Once the feature selection is complete, the preprocessed data is fed into an ML model, which is trained on labeled examples of normal and attack network flows. The trained model then classifies incoming network traffic as either benign or attack flows based on learned patterns and characteristics. Postprocessing techniques may be applied to the model's output to refine and filter detection results, reducing false positives and enhancing detection accuracy. Finally, the output of the ML-based NIDS includes alerts or notifications indicating potential network security breaches or suspicious activities within the network. This enables timely response and mitigation measures. In the following, we explain the required information concerning each of these modules utilized in this thesis.

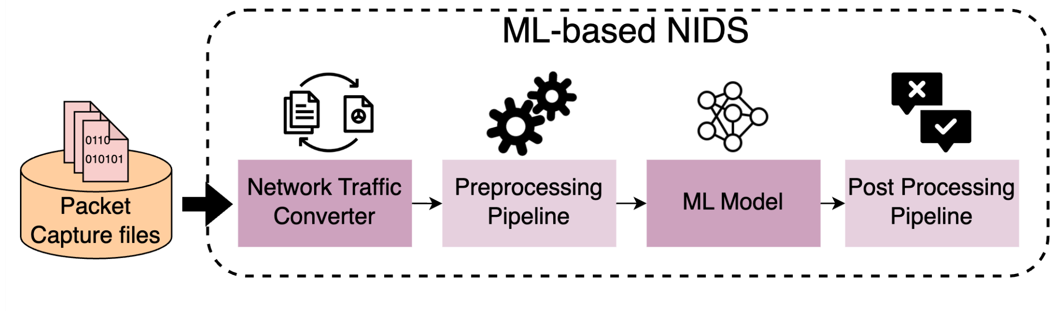

Figure 1: The architecture of a Machine Learning-based Network Intrusion Detection System (NIDS) is composed of various modules that collaborate to detect network intrusions. These modules include the Network Traffic Converter, Preprocessing Pipeline, ML Models, and Post Processing Pipeline. This work primarily contributes to the Preprocessing and ML Model modules.

2.2 NETWORK TRAFFIC DATA

To capture network traffic, packets can be intercepted as they move through a network interface [141]. This process involves selecting specific points within the network architecture, such as routers or firewalls, for capturing. Once a suitable point is identified, a capturing tool must be chosen and installed on a server connected to the capture point. This tool is then configured to capture traffic on the desired interface or VLAN. The data obtained from this captured network traffic can be in either packet-based or flow-based formats. Packet-based data typically contains comprehensive details about each individual packet, including source and destination IP addresses, port numbers, protocol type, packet size, timestamp, and payload data. In contrast, flow-based data is more aggregated and generally consists of metadata from network connections. This aggregation involves combining packets with shared properties, such as 5-tuple information, within a specific time window into one flow. Flows can be either unidirectional, aggregating packets from source A to destination B, or bidirectional, considering packets in both directions, from source A to destination B and vice versa.

2.2.1 *Public Network Traffic Datasets*

When training an ML model, it is crucial to utilize relevant data pertaining to the task at hand. Researchers often rely on various publicly available network traffic datasets to develop ML-based NIDS and evaluate the proposed models' abilities. However, these datasets employ different traffic converters, either packet-based or flow-based, leading to differences in feature extraction methods.

In this thesis, our goal is to comprehensively evaluate proposed models and algorithms across a range of diverse network traffic datasets. To accomplish this, we employed NFStream (as mentioned in Section 2.3) to extract flow features from publicly available network traffic files in PCAP format. Additionally, we utilized the provided information on attacker or victim IP addresses to label feature files for evaluation, which necessitated ground truth labels indicating whether the traffic was malicious or benign.

In Section A.2, we briefly explain CICIDS17, UNSW-NB, CTU-13, CICDoS, and Botnet network traffic datasets which are used in this thesis. Our decision to use these datasets was based on their availability of network traffic files in PCAP format and the presence of various attack types for thorough evaluation.

2.2.2 *Categories of Network Intrusions*

Various methods can be used to categorize attacks or malicious network patterns, such as those outlined in [70, 76, 152, 154]. One approach is based on the Open Systems Interconnection (OSI) model, which divides system communication into physical, Media Access Control (MAC), network, transport, and application layers. Different protocols and specifications are implemented at each of these layers. The Application layer employs Hypertext Transfer Protocol (HTTP) to provide web services, File Transfer Protocol (FTP) for large file transfers, and Simple Mail Transfer Protocol (SMTP) for sending electronic mail. The Transport layer utilizes Transmission Control Protocol (TCP) for dependable data delivery and User Datagram Protocol (UDP) to minimize protocol overhead. The Network layer facilitates data delivery using Internet Protocol (IP) addresses through IP and generates error messages using Internet Control Message Protocol (ICMP). In Wi-Fi networks, the MAC layer employs CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), while Additive Link On-line Hawaii Area (ALOHA) is utilized for military purposes. Finally, the Physical layer manages physical transmission characteristics.

In this thesis, we focus on attacks exploiting protocol vulnerabilities in network, transport, and application layers, which we briefly explain below.

Attacks on the network layer exploit weaknesses in IP and ICMP protocols, including hijacking, IP spoofing, and the Smurf attack [154]. IP spoofing disguises the attacker's identity by forging IP addresses, potentially causing network congestion and disruption. Hijacking involves seizing control of legitimate users' IP addresses and facilitating unauthorized access to confidential data. The Smurf attack inundates victims with ICMP packets, crippling network performance. Both TCP and UDP protocols in the transport layer are susceptible to security attacks such as TCP flooding and UDP flooding attacks [152]. TCP flooding inundates victim nodes with ICMP ping requests, delaying network connections [143]. UDP flooding overwhelms victim nodes with excessive UDP packets, rendering them unreachable [169]. Therefore, the flooding type of DoS and DDoS attacks can be categorized under the transport layer of the OSI model.

The HTTP, SMTP, and FTP protocols in the application layer are often targeted by botnets, posing significant network security threats. Botnets leverage malware such as Trojans, worms, and viruses to compromise systems and intercept sensitive data exchanged over these protocols [66]. Additionally, botnets exploit vulnerabilities like SQL injection in data-driven applications, enabling unauthorized access to websites.

FTP, commonly utilized for large file transfers, becomes a prime target for botnet attacks, including FTP bounce attacks aimed at hijacking data transfers. Similarly, botnets orchestrate SMTP attacks, utilizing worms and viruses for email spoofing and password sniffing, further amplifying the risks to network security. In addition to the mentioned attacks, many other attacks in these three layers use the protocol's vulnerabilities. Furthermore, the emergence of new types of attacks, such as Multi-Stage Attacks (MSA), involves sequential steps, where each step might not be necessarily recognized as an attack on its own, but their collective execution constitutes a comprehensive attack traffic [12]. The SlowDoS attacks and Botnets are well-known examples of MSA. Additionally, BruteForce attacks, where an attacker attempts to crack the password, are categorized as application layer attacks. These encompass capturing and altering data transmitted between the user and the system [124].

2.3 NETWORK TRAFFIC CONVERTER

In this thesis, the NIDS is composed of four main components: a traffic file converter, a preprocessing pipeline, an ML model, and a post-processing pipeline, as discussed in Section 2.1. To convert the traffic file to feature files and extract statistical features of a network flow, we have used NFStream [11], a renowned Python framework. NFStream can extract over 80 features, which are categorized into core features, statistical features, and post-mortem features. The features and their respective descriptions are listed in Section A.1.

It is important to note that certain features like source and destination IP addresses, source and destination MAC addresses, or port numbers can reveal certain information to the model. This might prevent the model from learning the statistical features and cause it to only differentiate between attack and benign flows based on their addresses [57]. Therefore, in this thesis, we excluded these specific features before training the model.

2.4 MACHINE LEARNING APPROACHES

In this thesis, we explore different types of ML models, each tailored to specific training strategies, annotated data requirements, and intended tasks. Here, we explain several approaches frequently employed.

2.4.1 *Supervised Learning Models*

Supervised ML models rely on extracting the relationship between features and labels to discern between normal and abnormal behavior [145]. As a result, these models require training data that includes the ground truth label for each flow. However, the scarcity of annotated datasets imposes limitations on the capacity of supervised learning models, restricting their ability to detect new flows that resemble those present in the training set [148].

Supervised learning models encompass various categories, including linear, instance-based, and tree-based models [159]. Each category offers unique strengths and weaknesses, catering to different types of data and problem domains. Additionally, advancements in deep learning have led to the development of complex neural network architectures capable of capturing intricate patterns in data [107]. Furthermore, ensemble methods like gradient boosting leverage a combination of multiple weak learners into a strong and robust predictive model, enhancing the overall classification performance [50]. In the following, some of the supervised learning models are briefly explained.

Logistic Regression (LR)

The primary objective of logistic regression (LR) is to model the probability that a given input belongs to a particular class [74]. Unlike linear regression, which forecasts continuous values, logistic regression employs the logistic function (sigmoid function) as depicted in Equation 1 to confine the output within the range of 0 and 1 [158]. Therefore, this model is suitable for binary classification tasks.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

In the Equation 1, z is the linear combination of input features and coefficients. The LR model is trained using a maximum likelihood estimation process [137]. The aim is to identify the set of coefficients that maximizes the likelihood of the observed data given the model. This optimization is usually accomplished using iterative algorithms such as gradient descent.

Decision Tree (DT)

A decision tree (DT) [21] is a predictive modeling tool that operates by recursively partitioning the dataset into subsets based on the most significant attribute at each node, ultimately leading to a tree-like structure where each leaf node represents a decision or an outcome. The algorithm employs a top-down approach, starting with the entire dataset and selecting the attribute that maximizes information gain or minimizes impurity, often measured using metrics like the Gini index, at each node, effectively dividing the data into homogeneous groups. This process persists until it reaches a predetermined stopping criterion, such as the maximum tree depth or the minimum number of samples in a leaf node.

Random Forest (RF)

Random Forest (RF) [22] is an ensemble learning technique that builds multiple DTs during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. The main difference between an RF model and a single DT lies in the approach to diversity and robustness. Therefore, this approach enhances the model's ability to handle complex patterns in network traffic data while mitigating the risk of overfitting and reducing sensitivity to variations. By averaging out individual tree biases, the RF model provides a more robust and accurate predictive model for identifying malicious activities in network traffic, making it an effective ML model for the ML-based NIDS [37].

Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) [142] is a type of artificial neural network designed for supervised learning that consists of multiple layers of interconnected nodes or neurons. It is characterized by an input layer, one or more hidden layers, and an output layer. Each connection between neurons is associated with a weight, and each node has an associated activation function. During training, the model learns optimal weights through the backpropagation process, where the error between the predicted output and the ground-truth target is minimized by adjusting the coefficients using gradient descent. The hidden layers enable the network to capture complex and non-linear relationships within the data. The activation functions introduce non-linearity, enabling the network to grasp and extract intricate patterns within the input data. MLPs have demonstrated effectiveness in numerous applications, including computer vision and natural language processing, owing to their ability to model intricate relationships within high-dimensional data [63].

Extreme Gradient Boosting (XGBoost)

XGBoost, an open-source machine learning library, offers an implementation of the gradient boosting algorithm [30]. Unlike the RF technique, gradient boosting is an ensemble learning method that combines several weak learning models, typically DTs, to create a more accurate predictive model [101]. However, XGBoost distinguishes itself by iteratively enhancing overall performance. Instead of training each DT with a subset of features, XGBoost sequentially adds new trees to the ensemble model, aiming to rectify errors made by preceding trees. This iterative refinement process results in better performance compared to RF classifiers, particularly evident in terms of training speed and scalability.

2.4.2 *Unsupervised Learning Models*

Unsupervised learning constitutes a fundamental machine learning paradigm designed to identify underlying patterns, structures, or relationships within unannotated datasets [40]. In unsupervised learning, the algorithm delves into the intrinsic structure within the data, rendering it suitable for tasks such as clustering, dimensionality reduction, outlier detection, and anomaly detection in an ML-based NIDS [9]. In contrast to supervised learning, which relies on labeled data with predefined outcomes, unsupervised learning operates on unlabeled datasets, necessitating algorithms to autonomously uncover latent patterns or groupings without explicit target labels [164]. In this section, we will provide a brief explanation of Autoencoders, which will be used as a baseline for one of the contributions of this thesis.

AutoEncoders

Autoencoders represent a neural network architecture employed in unsupervised learning endeavors, particularly for tasks involving dimensionality reduction and feature learning [166, 175]. The fundamental concept of autoencoders involves encoding input data into a lower-dimensional representation (referred to as the encoding or bottleneck layer), followed by decoding it back to the original input [27]. During training, the network learns to reconstruct the input as accurately as possible, forcing the model to capture essential features and patterns within the data. Autoencoders consist of an encoder, which transforms the input data into a latent space representation, and a decoder, which reconstructs the input from the encoded representation.

In the context of NIDS, autoencoders can be valuable for anomaly detection [175]. By training an autoencoder on a dataset containing normal network behavior, the model learns to encode the typical patterns and structures within the data [33, 46]. During deployment, the autoencoder is applied to new, unseen data. If the reconstructed output significantly deviates from the original input, it indicates an anomaly or potentially malicious activity. Therefore, autoencoders serve as an effective unsupervised model applicable to ML-based NIDS.

2.4.3 *Self-Supervised Learning Models*

Self-supervised learning (SSL) is a machine learning paradigm that serves as a bridge between supervised and unsupervised learning by harnessing the intrinsic structure present within the data to generate labels [128]. In supervised learning, models are trained on annotated datasets, which can be costly and time-consuming to acquire, especially for domains like network security and anomaly detection [67]. On the other hand, unsupervised learning relies on extracting patterns from unlabeled data, which might not capture the nuanced and complex nature of network behaviors effectively [128].

SSL addresses these challenges by formulating tasks that generate pseudo-labels from the input data. Instead of relying on external annotations, the model creates its own labels, often through pretext tasks [108]. For NIDS, pretext tasks can involve predicting the next step in a network sequence, identifying whether two network flows belong to the same category (contrastive learning), or reconstructing parts of the network traffic that was corrupted [4, 119, 170, 176]. These pretext tasks guide the learning process and enable the model to capture essential features and relationships within the data without requiring explicit labels.

One advantage of self-supervised learning over pure unsupervised learning in the context of anomaly detection is that it introduces structured learning by defining pretext tasks that inherently encode relevant information about network behavior. While unsupervised learning methods like clustering and dimensionality reduction aim to uncover latent patterns, SSL explicitly defines tasks that guide the model to learn more meaningful representations. This structured approach often results in better performance, especially when dealing with complex and dynamic network

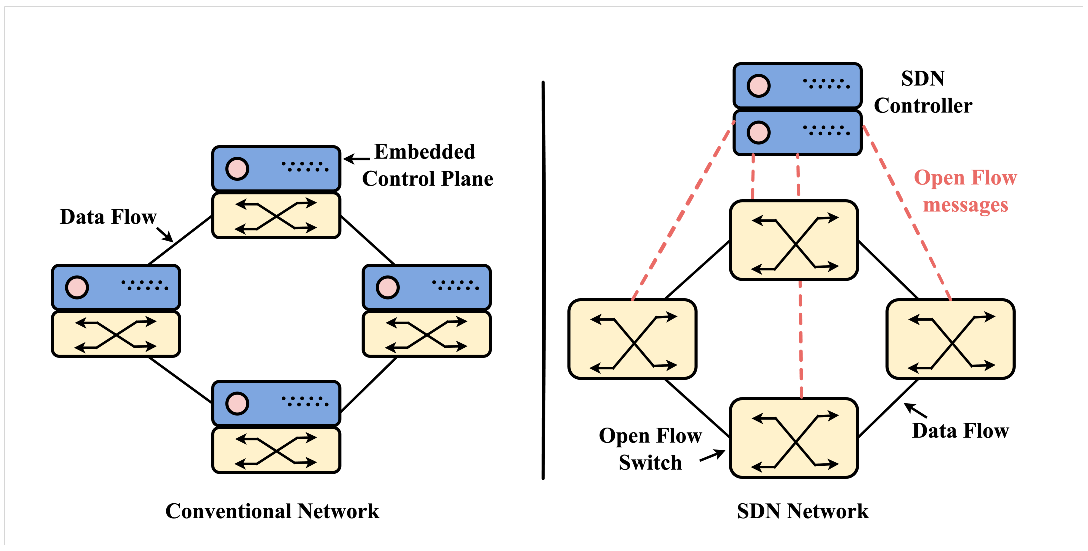

Figure 2: A general architecture of a conventional network and Software-Defined Networking (The Figure is inspired by [144]). As depicted, SDN separates the control plane from the data plane, making network management more flexible.

environments where anomalies may manifest in subtle ways. Consequently, it can improve the generalization performance of the ML-based NIDS. By integrating SSL into NIDS, we enhance the model's ability to discern between normal and malicious network behavior, contributing to more robust and effective intrusion detection systems.

2.5 SOFTWARE-DEFINED NETWORKING

The key fundamental concept of SDN is decoupling the control plane from the data plane and enabling the network control with high programmability [127, 153]. Figure 2 (inspired by [144]) illustrates a comparison between conventional network architecture and SDN Network. In SDN, the responsibility of forwarding network traffic is with the data plane, while controlling the network is the control plane's responsibility. Accommodating a logically centralized control plane gives it a global view, enabling SDN to support different applications in the network. The most popular protocol in SDN to make communication between the control plane and the data plane is OpenFlow protocol [78].

OpenFlow allows dynamic programming of network traffic management through predefined match rules set by the SDN controller. Flow tables in OpenFlow switches contain entries comprising match fields, priority, instructions, and timeouts. These entries are populated by the controller and used to process incoming packets [72]. Packet processing involves comparing packet headers with match fields in flow entries to update action sets and forward packets accordingly [13]. This approach enhances switch behavior customization, offering greater network management flexibility and reducing device complexity [36, 93].

The northbound API simplifies network management for SDN applications by providing a high-level view of the physical network, freeing operators from hardware intricacies [135]. These policies are transmitted to the data plane through the controller using both northbound and southbound APIs. SDN applications cover diverse areas such as traffic engineering for dynamic rerouting, network security (the focus of this thesis), Quality of Service (QoS), etc. [138].

2.5.1 *P4 Language*

The high-level language for Programming Protocol-Independent Packet Process (P4) has emerged to provide a flexible and efficient means of defining how network packets are processed and forwarded within SDN architectures [92]. P4 aims to enable programmable forwarding and foster innovation in networking protocols while promoting hardware independence by abstracting underlying hardware details [88]. Figure 3 illustrates the packet processing pipeline in a programmable switch using the P4 language [118]. Upon packet arrival, the parser extracts predefined header field values [18]. These values then enter the ingress match-action pipeline, driven by match-action tables populated by the controller. Table entries match keys and execute corresponding actions. The order of table execution is defined within a control block. Following the ingress pipeline, packets may be forwarded, dropped, replicated, or sent to the controller. The egress pipeline handles per-instance modifications like queueing and scheduling, while the deparser reassembles modified headers into outgoing packets [19, 54].

The P4 language facilitates value concatenation and slicing, storage and updating using registers, and tracking network traffic statistics with counters [73]. However, P4 lacks support for mathematical operations like division, logarithm, and exponentiation, as well as traditional loop constructs such as 'for' or 'while' loops, making it challenging to implement complex algorithms requiring these operations [171].

2.5.2 *SDN Security Challenges*

While SDN provides centralized enforcement of security policies and a comprehensive network overview, simplifying attack detection compared to conventional networks, its centralized architecture also introduces inherent security challenges that require attention [35, 41, 72, 83, 147]. For instance, attacks targeting the control plane, such as packet-in flooding and controller's switch table flooding, pose significant threats [83]. In packet-in flooding, adversaries overwhelm the controller with malformed network packets, exhausting its computational resources. Similarly, the controller's switch table flooding exploits the lack of authentication in OpenFlow control packet reception, allowing attackers to flood the controller's switch table with fake entries, degrading its performance over time. A compromise of the controller could have severe consequences, leading to widespread network disruption or unauthorized access to sensitive data.

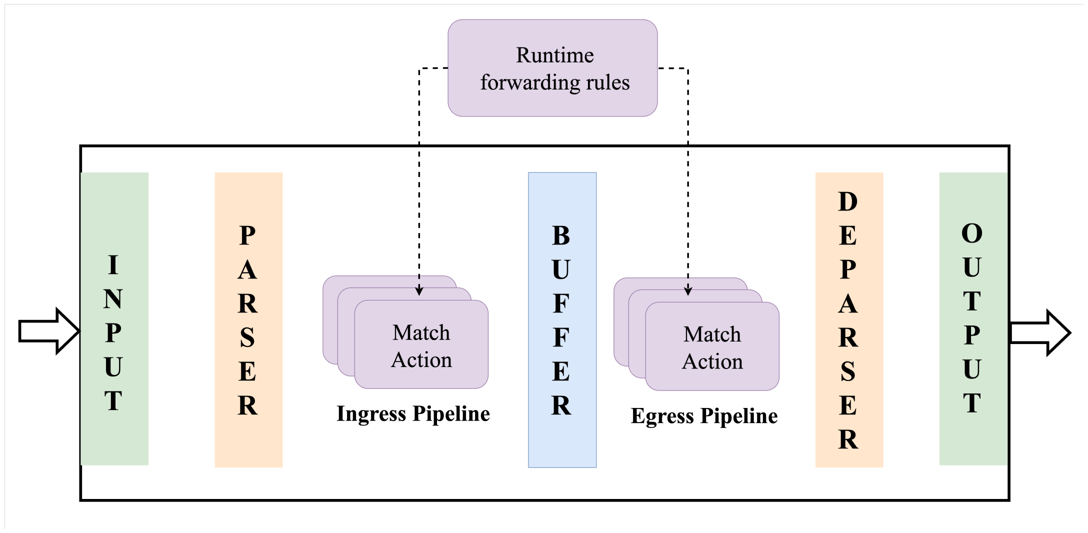

Figure 3: The P4 processing pipeline consists of several stages (The Figure is inspired by [118]). Each stage plays a crucial role in processing network packets, from parsing incoming data to preparing it for transmission. This modular approach enables efficient and customizable packet processing tailored to specific networking requirements.

Therefore, It is essential to deploy an NIDS in SDN to overcome the security challenges associated with SDN. Anomaly-based NIDSs can detect previously unseen network attacks in SDN networks, similar to conventional networks when compared to signature-based NIDS. The following section explains the related work on anomaly-based NIDSs deployed in SDN.

2.6 RELATED WORK

This section delves into research relevant to each contribution made in this thesis. Firstly, we delve into the assessment of generalization in ML models, encompassing both supervised and unsupervised approaches. Subsequently, we discuss pertinent studies concerning the deployment of ML models in SDN environments.

2.6.1 *Generalization of Machine Learning Approaches*

Generalization performance in the context of ML-based NIDS pertains to the ML model's ability to detect anomalies of data it hasn't seen before. In this thesis, when the new unseen data belongs to the same dataset used for training, we refer to it as *intra-dataset* generalization performance. Conversely, if the new unseen data comes from different datasets, potentially with different distributions, we refer to it as *inter-dataset* generalization performance.

Generalization in Supervised Learning Models

In this section, we delve into related works that have investigated the generalization performance of supervised learning ML-based NIDS. Subsequently, we present related studies that developed feature selection algorithms. Feature selection is highlighted as a method aimed at enhancing the intra-dataset generalization performance of supervised learning ML-based NIDS.

In [39], the authors delved into the intra-dataset and inter-dataset generalization of supervised learning ML-based NIDS. They created a pipeline that encompassed twelve supervised learning models spanning various families. These models were employed to evaluate detection performance across common attack types (such as DoS, SSL, and botnet) originating from the same dataset (intra-dataset) and two distinct datasets (inter-dataset). Notably, tree-based models exhibited superior generalization performance in the inter-training-dataset scenario. However, they encountered challenges in achieving robust detection performance when faced with unseen data from different datasets. The study also observed that ensemble ML models got better detection performance for intra-dataset, attributed to their capacity to derive detection results through a combination of models. In [38], the authors stressed the importance of evaluating the generalization performance of ML-based NIDS. They highlighted the necessity to improve it, especially after finding that while some state-of-the-art models showed strong detection performance on unseen data from the training dataset (intra-dataset generalization), they struggled when faced with data from different sources (inter-dataset generalization). The study focused on the top three models with exceptional intra-dataset generalization. It found that for network-centric attack classes like brute force and denial of service, these models maintained high precision and recall, with losses below 5%. However, performance varied across other attack classes, ranging from significant recall losses for botnets to complete degradation of precision for web attacks and infiltrations. These findings underscored the complexities involved in achieving robust generalization across diverse attack types in ML-based NIDS. Building upon the insights from [39] and [38], [95] delved into the generalizability of ML-based NIDS using seven supervised and unsupervised models across four well-known network datasets. Their investigation revealed that none of the models could generalize across all datasets effectively. Notably, they highlighted a high degree of asymmetry in generalizability, where swapping the source and target domains significantly altered classification performance. The study found that unsupervised models exhibited superior generalization performance compared to supervised learning models. Furthermore, the authors employed SHAP [112], an explainable feature importance extraction method, to delve into the reasons behind the lack of generalizability. Their analysis revealed that the main factor contributing to this issue was the strong correlation between the values of one or more features and the Attack/Benign classes in certain network traffic dataset-model combinations, which were absent in other network traffic datasets with different feature distributions. These findings shed light on the intricate dynamics affecting the generalization capabilities of ML-based NIDS and underscored the importance of robust feature representation across diverse datasets.

Flow Feature Selection in ML-based NIDS

The findings from [38, 95] highlight the considerable influence of feature selection on the generalization of supervised models, particularly when employing ensemble methods comprising multiple models. In [48], the authors leveraged Spearman's rank correlation coefficient to identify unused features and reduce the feature dimension. Furthermore, they devised an ensemble learning approach combining logistic regression (LR), decision trees (DT), and gradient boosting to harness the strengths of each ML model. They conducted a comparison between this ensemble model and seven standalone ML models, both with and without employing a feature selection approach. The results indicated improved performance for the ensemble ML model when feature selection was employed. In [7], the authors explored multi-class ML-based NIDSs across four benchmarking datasets using six ML models. They emphasized the importance of a robust preprocessing pipeline for enhancing detection performance, including data cleaning, transformation, normalization, and feature selection. Their study addressed critical questions about dataset selection criteria for maintaining quality in IoT-based IDSs, the impact of quality assurance factors on ML-based IDSs in IoT contexts, and how quality assurance methods can boost performance. They also proposed a lightweight framework based on model quality assurance methods to improve IDS detection rates in IoT environments. In [80], the authors introduced a novel multi-stage optimized ML-based NIDS aimed at reducing computational complexity while preserving detection performance. The study explored the impact of oversampling techniques, particularly SMOTE [28], on training sample sizes, resulting in a significant reduction (39-74%) compared to original datasets. Additionally, two feature selection methods, Information Gain (IGBFS) and Correlation-Based Feature Selection (CBFS), were evaluated, both reducing the feature set size by almost 60% and further decreasing the required training sample size (33-50% reduction post-SMOTE). Moreover, different ML hyperparameter optimization techniques were investigated, leading to improvements in detection performance, reduced computational complexity, and shorter detection times. Authors in [130] introduced a filter method for feature selection named EMFFS. Their approach involved aggregating the outputs of four filter selection methods, namely Information Gain, Gain Ratio, Chi-squared, and ReliefF, to score the features. By employing majority voting and a predefined threshold, they selected only 30% of the features. Their evaluation was limited to a single dataset, where they observed an improvement compared to scenarios without feature dimension reduction. They inferred that this enhancement indicated the presence of redundancy or non-informative features in the original dataset. Following [130], authors in [90] integrated Information Gain and correlation filter techniques to diminish feature dimensionality, focusing specifically on DDoS attacks. They divided the output of each individual feature selection method into three subsets using predefined thresholds. Subsequently, they combined the most relevant features from the union into the highest subset and the most relevant features from the intersection into the second-highest subset, resulting in a reduced set of features. Their findings demonstrated an enhancement in detection performance with the utilization of their feature selection algorithm.

Considering all the aforementioned related work underscores the critical importance of generalization performance, encompassing both inter- and intra-dataset scenarios, as a crucial metric when utilizing ML-based NIDS. However, supervised learning models exhibit poor generalization performance as they aim to establish relationships between features and labels within a single dataset. The mentioned related works indicate that proper feature selection can enhance detection and generalization performance, particularly in the context of intra-dataset generalization and, to some extent, inter-dataset generalization. Despite the development of ensemble feature selection methods using various statistical approaches, many overlook the significant issue of dataset imbalance in network traffic datasets. These methods typically apply feature selection once on the entire dataset to derive a subset of features. Consequently, the feature selection process tends to prioritize learning features relevant to the majority class while disregarding minority ones. This oversight can adversely affect the generalization performance.

Generalization in Unsupervised Learning Models

In [126], the authors introduced TS-IDS, a groundbreaking graph-based Traffic-aware Self-supervised model for Network Intrusion Detection. TS-IDS revolutionizes intrusion detection by harnessing Self-Supervised Learning (SSL) techniques, which are integrated into the model to enrich the graph representation. Unlike traditional approaches, TS-IDS doesn't rely on predefined feature extraction methods. Instead, it leverages Graph Neural Networks (GNNs) specifically tailored for graph-structured data to capture intricate network relationships effectively. Their proposed traffic-aware SSL module can enhance the graph representation and consider the historical behavior of nodes and communication patterns between them (edge features). Node auxiliary labels are extracted based on traffic volume, labeling nodes with high traffic as suspicious. However, this approach is limited to detecting only volumetric attacks. In [106], the authors developed an ensemble feature selection approach integrated with contrastive learning. This approach extends the use of Autoencoder from unsupervised to supervised learning, allowing both normal visits and attacks to contribute to feature learning. Additionally, classical clustering algorithms were implemented under the supervised learning scheme based on network embedding of data labels. To combat imbalanced data distribution, the framework employed a modified version of Focal Loss [102], traditionally used in the image field, resulting in improved model performance for both binary and multi-class classification tasks. In [110], the authors proposed a contrastive learning-based approach that integrated flow labels into the embedding space along with flow features to facilitate flow classification. In this case, the sample features should be close to the correctly annotated samples (positive pair) while keeping the distance with the other clusters (negative pairs) high. Therefore, ground-truth labels are required to create the positive and negative pairs. They evaluated their approach in noisy and unbalanced datasets to show the effectiveness of their method. Similarly, [105] employed a supervised contrastive learning approach by incorporating sample labels to tackle imbalanced and constrained feature extraction capability issues. In this process, they dropped some layers in the encoder randomly to generate augmented data. Additionally, ground-truth labels were utilized to categorize the augmented samples.

The authors in [177] aimed to address the scarcity of annotated samples in network traffic by designing a self-supervised contrastive learning approach. They devised a heuristic method to construct contrastive tasks based on random masking of network packet sequences, generating positive and negative sample pairs to represent sample relationships effectively. Additionally, they proposed a contrastive cross-entropy loss function that combines supervised contrastive loss and cross-entropy loss, facilitating accurate decision-making while minimizing intra-class distance and maximizing inter-class distance. However, as explained in [61], important information can be retrieved from the initial packets; therefore, corrupting them may lead to information loss. Also, it should be noted that this approach was trained using both benign and attack flows. The authors in [167] introduced a self-supervised contrastive learning model that transformed the flow array into a two-dimensional array format resembling a gray image. Various image-related augmentations, including horizontal and vertical flips, random cropping, and shuffling, were applied to create positive and negative pairs. However, the use of image augmentations on sequential intrinsic flow data points might generate pairs with differing semantics. Additionally, the study utilized the attention mechanism as the backbone of the encoder. This mechanism autonomously learns and assesses the input data's contribution to output classification. It effectively suppresses less relevant features while boosting those crucial for classification in intrusion detection data. The method's generalization performance was evaluated across three diverse datasets. Similar to [126], authors in [26] extracted features from each node using GNN. However, their methodology diverged as they trained the SSL model in an unsupervised manner, devoid of label information. To generate positive pairs, they adopted a strategy involving the random selection of K neighbors for each graph node. Notably, this approach entailed masking information from certain neighbors of each node in the graph. Additionally, the authors conducted evaluations on two distinct datasets to demonstrate the generalization capabilities of their work.

2.6.2 *Machine Learning-based Intrusion Detection in SDN*

In this section, we present an overview of anomaly-based NIDS deployed in the SDN control or data plane and briefly explain the associated challenges. Deploying ML models in the SDN control plane can be a suitable option, as it has the required computational resources.

Authors in [49] developed HFS-LGBM IDS, which was deployed in the SDN control plane to detect network intrusions. To improve the detection performance, the authors used a hybrid feature selection algorithm which involved correlation-based feature selection and random forest recursive feature elimination. Then, they trained a LightGBM algorithm on the optimal feature subset obtained from the hybrid feature selection method. The comparative analysis with some single ML model baselines indicated improvements in the system's approach. In [97], authors improved the ML-based NIDS deployed in the SDN control plane. They addressed various challenges that ML models faced in achieving high detection performance, such as an imbalanced dataset, a proper feature selection method, and detection speed. They used conditional Generative Adversarial Networks (GAN) to address the imbalance dataset issue while using deep sparse autoencoder to reduce the feature dimension to the most relevant ones. Furthermore, they utilized NetFPGA to accelerate the packet processing task and increase detection speed. In [51], authors investigated detection performance for an ensemble method consisting of multiple ML models, including Support Vector Machine, Logistic Regression, Neural Networks, and Random Forests. In their proposed ensemble methods, each of these single models is trained on a different subset of features. In their evaluation, they illustrated how different feature sets can modify detection performance; therefore, they highlighted the feature selection method importance.

The mentioned research studies have mainly focused on improving the detection performance of ML-based NIDS deployed in the control plane. However, deploying such systems in the control plane can increase the risk of successful attacks against it. This is because the flows need to be forwarded to the control plane, which can also lead to network overload and overwhelm the control plane. In addition, the detection speed is not in the line rate. To overcome these challenges, some studies propose deploying ML-based NIDS on the programmable data plane.

One category of related work developed an external ML-based NIDS that communicates directly with a programmable P4 switch. Consequently, in these studies, the responsibility for feature extraction falls on the P4 switch, while the classification decision is handled by the external ML-based NIDS. In [123], authors employed various ML models, including RF, KNN, and SVM, within an external ML-based NIDS to identify TCP flood attacks. They introduced five distinct window-based flow features computed by the P4 switch, which were utilized for training the ML models. These features were derived by aggregating the values of individual packets over a predefined window duration. Similar to [123], in [24], multiple ML models are deployed in an external ML-based NIDS. However, they opted to compute flow-based features within a specific timeframe directly within the P4 switch rather than utilizing window-based features. Consequently, the input data for training the model was organized based on flows within a predetermined observation window size. Additionally, their proposed approach extended to the detection of a broader range of attacks. In these studies, the necessity to forward packets or flow features to an external ML-based NIDS can lead to heightened network load and latency.

The second category of related work embedded ML models directly inside the P4 programmable switch to construct an in-network ML-based NIDS. In [173], four ML models, including DT, SVM, NB, and K-means, were deployed on both software and hardware programmable switches (BMv2 and NetFPGA [183]). However, due to limitations in the mathematical operations supported by the P4 language, they were unable to compute statistical features of flows within the switch. Therefore, they utilized look-up tables to retrieve the specific results. Their investigation revealed that this approach required extensive table entries, resulting in heightened memory usage. Authors in [171] argued that tree-based ML models are the most suitable for deployment in P4 switches due to their compatibility with the match-action table structure. They asserted that other ML models like SVM, ANN, KNN, and NB are impractical in P4 switches due to the lack of support for certain operations in the P4 language. They trained a DT model using both packet-based and flow-based datasets, finding that while packet-based methods require fewer computational resources, their detection performance is inferior to feature-based DT models. Additionally, the flow-based implementation exhibited slower detection speeds due to the need to extract all packets of a flow. Moreover, hard-coding the DT model in the programmable switch reduces the flexibility of the In-Network NIDS, necessitating rewriting if the model is updated. Building on this research, authors in [98] implemented an RF model with 11 trees directly within the P4 switch (BMv2). In their approach, each flow feature was stored in a separate register based on its flow ID. However, this storage method resulted in increased detection time, as each register needed to be read and written repeatedly upon receiving new packets in the switch. Additionally, this framework may face compatibility issues with hardware-based switches. Authors in [23] expanded the concept of [98] by deploying multiple RF models for an In-Network NIDS. They introduced subflow features, which are derived from a specific number of initial packets rather than capturing all packets within a flow, aiming to enhance detection speed. In their framework, flows were divided into subflows, with individual RF models trained on each subflow. Additionally, they utilized bit concatenation to convert flow feature values into binary strings, thereby reducing the total number of required registers.

Deploying ML-based NIDS directly within programmable switches, like P4 switches, addresses the challenges of deploying ML-based NIDS in the control plane. However, their detection performance may not match that of NIDS deployed in the control plane, as programmable switches can only accommodate lightweight ML models. Thus, there's a need to develop a framework that leverages the advantages of deployment in both the control and data planes.

2.7 SUMMARY AND IDENTIFIED RESEARCH GAP

In this section, we delve into the diverse categories of ML models and address the critical considerations for employing ML-based NIDS. A significant advantage of ML-based NIDS lies in their capability to detect novel, unseen flows. Hence, it's imperative to evaluate their detection and generalization performance across varied scenarios. Generalization within ML-based NIDS can be delineated into two facets: within the same dataset (intra-dataset) and across different datasets (inter-dataset).

Diminishing the feature dimension to pertinent features can enhance intra-dataset generalization, especially within a supervised learning framework. However, several feature selection methods were used on all datasets. Given the highly imbalanced nature of network traffic datasets, they often prioritize features relevant to the majority class, neglecting informative features for minority ones. Furthermore, attacks can vary in behavior and distribution depending on the network traffic dataset. To tackle this challenge, we propose a novel approach to feature selection termed ensemble feature selection. This algorithm is constructed to identify a subset of features that offer valuable information across all classes present in the dataset. Additionally, it enhances the ability of supervised ML-based NIDS to generalize within the dataset. To address the constraints posed by annotated network traffic datasets and to achieve broader generalization across different datasets in machine learning-based NIDS, it is crucial to develop a generic representation of benign traffic through unsupervised learning. To this end, we propose a Self-Supervised Learning (SSL) approach combined with a novel augmentation algorithm specifically tailored for network traffic samples. These contributions pave the way for a more adaptable ML-based NIDS capable of detecting new, previously unseen, and out-of-distribution attacks.

Additionally, in this chapter, we explore network security challenges in SDN and deploying ML-based NIDS. Prior research suggests deploying ML models on the control plane due to its ample computational resources. However, this necessitates forwarding flows to the control plane for detection, potentially increasing attack vulnerabilities, causing detection delays, and overloading the network. To tackle these issues, some approaches in the literature deploy ML models on the data plane, but this may reduce detection performance due to limited computational resources. To overcome these challenges, we propose a framework for deploying ML models in both the data plane and the control plane. The data plane handles intrusion detection unless confidence falls below a threshold, in which case it is forwarded to the control plane. This contribution increases detection performance of the ML-based NIDS in the SDN while reducing network overload, and detection delay.

3 ENSEMBLE FLOW FEATURES SELECTION

Machine Learning (ML)-based Network Intrusion Detection Systems (NIDS) have emerged as a critical component in modern cybersecurity infrastructure to detect network intrusions before they damage the network. Supervised learning models (mentioned in Section 2.4.1) are one of the main categories of ML models that are used in anomaly-based NIDSs. These ML models can train on various network traffic datasets to learn the statistical flow features and distinguish between benign and attack flows. Supervised ML-based NIDSs are trained on annotated datasets, which makes them interpretable and helpful for understanding the decision-making procedure [15]. By continuously analyzing network traffic patterns, these ML-based NIDSs can proactively detect potential threats, enhancing overall network security and network management [20].

The effectiveness of ML-based NIDS detection can be influenced by the informativeness of its flow features [3, 75]. However, with the vast amount of data generated in network traffic and different types of traffic patterns, not all flow features are equally important for detecting network intrusions [91, 116].

Definition — *Feature selection techniques* help to identify the most informative and discriminative features, reducing computational complexity and improving model performance [65, 160, 178].

Additionally, feature selection helps to optimize model generalization and interpretability, which leads to better understanding and management of security threats in complex network environments [20]. Therefore, integrating feature selection mechanisms is an essential preprocessing step, which can enhance the efficacy and efficiency of ML-based NIDS [62].

While feature selection offers numerous advantages, it is crucial to carefully choose informative features to effectively cover all network traffic types. Network traffic datasets often exhibit imbalances, with benign flows making up the majority of data. Additionally, among samples of attack flows, certain types of attacks may be more prevalent than others. Therefore, the feature selection approach should not be biased towards the predominant samples. Additionally, given the intrinsic heterogeneity of network traffic and the emergence of new traffic patterns, it is essential to select a set of transferable features. Training ML models with these features should indeed empower the model to achieve high detection performance with a high degree of generalizability.

Definition — *Transferable features* refer to a set of network traffic features that can be effectively used for training ML models across different network traffic datasets or scenarios, regardless of whether they exhibit similar or diverse network traffic patterns.

However, selecting flow features from only one network traffic dataset can diminish the transferability of the feature to different network traffic datasets with varying patterns.

To address these challenges, this chapter proposes a preprocessing and feature selection pipeline designed to minimize noise in the feature space while preserving informative features. This approach aims to decrease the complexity of ML models and reduce training time while maintaining high detection performance. Moreover, the proposed method incorporates a data-driven approach to extract transferable network traffic features across diverse traffic patterns.

3.1 EXPLORATORY ANALYSIS OF NETWORK TRAFFIC DATASETS

In order to develop an appropriate ML model, it is crucial to gain a thorough understanding of network traffic datasets through exploratory data analysis (EDA). In the following sections, we conduct various analyses to comprehend the flow features and distribution of the network traffic datasets.

3.1.1 *Benign and Attack Samples Distribution*

In this thesis, we utilize five distinct network traffic datasets, each containing various types of attacks. Detailed information about the data collection process for each dataset is provided in Section A.2. After utilizing the NFstream tool [11] to convert the pcap traffic flows into the feature files, 88 flow features are extracted for each network traffic dataset. These features are available in Section A.3. After removing redundant flow samples from network traffic datasets, the respective sample counts are as follows: CICIDS17 contains 709,517 samples, SlowDoS contains 198,414 samples, CTU13 contains 2,398,949 samples, Botnet contains 205,611 samples, and UNSW-NB contains 237,853 samples. Figure 4 illustrates the distribution of benign and attack flows within each dataset. As depicted, most datasets (except the Botnet dataset) exhibit highly imbalanced distributions, with the majority class of benign flows. This imbalance is a critical consideration, as it can impact the detection performance of ML models, potentially leading to overfitting [79]. Awareness of imbalance issues in network traffic datasets is essential for ensuring that the feature selection algorithm and subsequent model training are not unduly affected by the disproportionate class distribution.

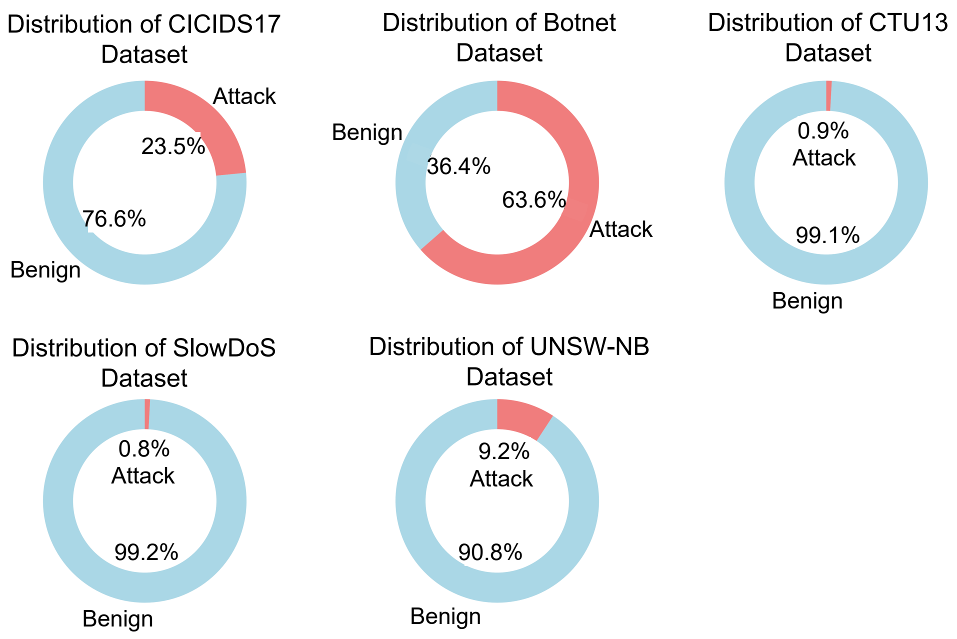

Figure 4: The distribution of attack and benign flow samples within five distinct network traffic datasets. It demonstrates that most of the network traffic datasets are predominantly imbalanced, with benign flows comprising the majority of samples in most cases.

3.1.2 *Flow Features Prunning*

Each statistical feature within a network traffic dataset provides information for distinguishing between benign and attack flows. However, it's crucial to prevent potential information leakage as this could introduce bias into the ML model and hinder its ability to learn traffic patterns accurately. For example, IP address information may exhibit a high correlation with attack or benign labels due to specific victim or attacker hosts in the network. Training the model with such information could bias it to merely learn associations between IP addresses and labels rather than capturing the underlying patterns of attacks. Additionally, certain features, such as time-related ones (excluding duration), may be heterogeneous across different network architectures, potentially hindering the ML model's ability to generalize well.

To address these issues, a preprocessing step is undertaken. In this step, all five-tuple features, including source and destination IP addresses, source and destination MAC addresses, and port numbers, are removed. Furthermore, all time-related features are eliminated from the dataset. These eliminations result in reducing the feature set from 88 to 50 in each network traffic dataset, ensuring that the model is trained on a more generalized set of features.

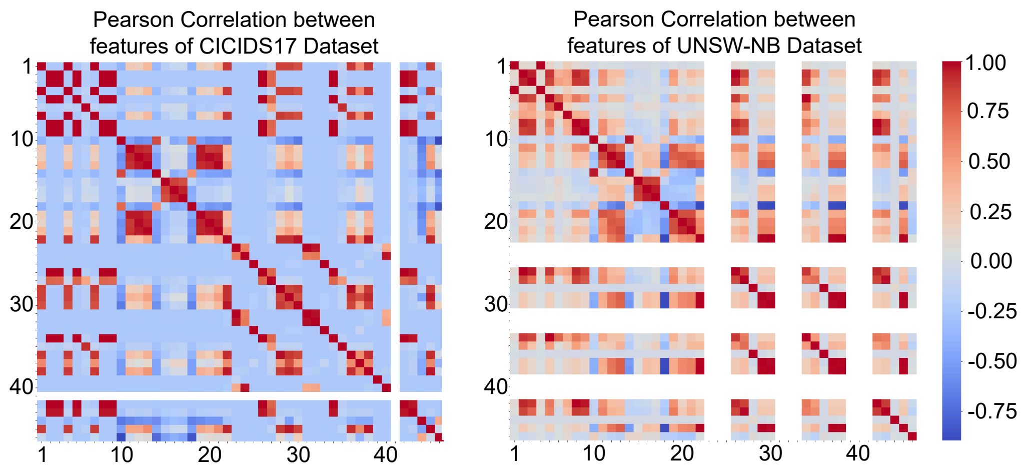

Figure 5: Pearson correlation between flow features within the CICIDS17 and UNSW-NB datasets. Red points represent direct linear correlations between features, while blue points indicate indirect correlations. As depicted, each network traffic dataset can have different feature correlations. The list of features with their corresponding identity numbers can be found in Section A.3.

3.1.3 *Pearson Correlation between Flow Features*

In this step, we investigate the Pearson correlation between features to understand their linear relationships. The Pearson correlation coefficient measures the strength and direction of linear relationships between variables (here, they are flow features), indicating how changes in one variable are associated with systematic changes in another [146]. Equation 2 shows how to calculate Pearson correlation between two variables X and Y [146].

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2}\sqrt{\sum(Y_i - \bar{Y})^2} \quad (2)$$

where X_i and Y_i represent individual data points corresponding to variables X and Y, respectively. Similarly, $\bar{X}$ and $\bar{Y}$ denote the average values of variables X and Y, respectively.

Figure 5 displays the Pearson correlation among features within the CICIDS17 and UNSW-NB datasets, accompanied by heatmap plots. Additional feature correlation figures for other network traffic datasets are provided in Section A.5. Furthermore, to enhance readability, the figure includes feature identity numbers, while a comprehensive list of features with their corresponding identity numbers can be found in Section A.3. High linear correlations between features suggest redundancy, which can be addressed by removing such features to reduce dimensionality, simplify models, and improve computational efficiency without significant loss of information [44]. This process can be integrated into a feature selection algorithm aimed at mitigating multicollinearity issues among features, consequently ensuring more stable parameter estimation by ML models [85]. Correlation analysis provides valuable insights into the underlying structure of data, uncovering patterns or dependencies that may not be immediately evident. This aids in data preprocessing by identifying and addressing correlated features, ensuring that the data is cleaned and prepared for analysis. By eliminating redundant or irrelevant information, ML models can be trained without being influenced by extraneous factors.

In Figure 5, the white line represents zero variance features, indicating those with only one value. Such features are considered non-informative as they do not contribute to distinguishing between different data points.

Additionally, the comparison between Figure 5 and the corresponding correlation plots for other network traffic datasets in Section A.5 highlights the notable differences among these datasets and their respective attack patterns. Such disparities pose challenges for feature selection methodologies.

3.1.4 *Low Variance Filtering*

Low variance statistical method for feature selection involves identifying and removing features with very little variation across various network traffic datasets [17]. Features with low variance have nearly constant values and are often considered non-informative. By eliminating these low variance features, the dimensionality of the dataset can be reduced, simplifying models and improving computational efficiency. Moreover, removing such features can help mitigate the risk of overfitting, as the model is less likely to memorize noise or irrelevant information. Low variance feature selection is particularly useful in scenarios where datasets contain numerous features, and there's a need to prioritize those that contribute significantly to the predictive performance of the model. Upon examining various correlation figures, including Figure 5 and those in Section A.5, it becomes evident that different features have zero variance within these datasets. This observation highlights the differences among network traffic datasets and demonstrates that distinct features may be informative in detecting various types of attacks.

3.1.5 *Flow Features Distribution*

In EDA, understanding the distribution of features can provide fundamental insights into the underlying structure and characteristics of the dataset. By visualizing feature distributions using tools like boxplots, it is possible to gain a comprehensive understanding of the data's central tendency, variability, and potential outliers [151]. This understanding is essential for identifying patterns, trends, and anomalies within the dataset, which in turn informs subsequent analysis and modeling decisions.

For instance, Figure 6 illustrates the distribution of certain features within the UNSW-NB network traffic dataset. It is evident that the distribution between attack and benign flows within this dataset varies.

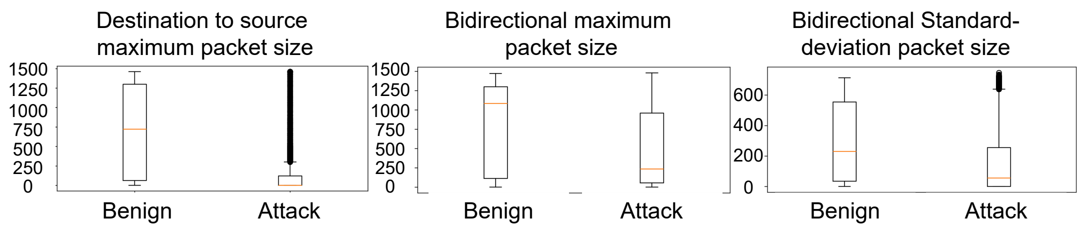

Figure 6: The disparities in the distribution of three chosen features between attack flows, and benign flows within the UNSW-NB network traffic dataset highlight clear distinctions. These differences serve to identify these features as potentially informative for distinguishing between attack and benign flows.

3.2 FEATURE SELECTION APPROACHES

EDA of five different network traffic datasets reveals their diversity, influenced by differences in network intrusion types, architectures, and management rules. This variability extends to general network traffic behavior, which is susceptible to concept drift and the emergence of new intrusions. To mitigate the effects of concept drift, feature selection approaches can be employed. These methods aim to identify the most relevant features while mitigating the influence of those that do not accurately represent traffic patterns. By reducing the complexity of machine learning models while maintaining high detection performance, these approaches enable effective adaptation to evolving network dynamics.

In this section, different feature selection approaches that can be used to select relevant flow features are introduced. These methods are used in our proposed Ensemble feature selection approach.

3.2.1 *Random Forest Gini Importance*

Random Forest's Gini Importance can serve as a feature selection approach. It operates by assessing the significance of each feature in a random forest model through the reduction of Gini impurity, a measure quantifying the degree of disorder within a dataset [111]. Mathematically, Gini impurity $(\mathtt{Gini}(p))$ for a node t with probability distribution p across K classes is calculated as:

$$\text{Gini}(p) = 1 - \sum_{k=1}^{K} p_k^2 \tag{3}$$

During the construction of decision trees within the random forest, at each node, the algorithm evaluates different features to determine optimal splits. The decrease in Gini impurity resulting from each feature's split is recorded and averaged across all trees, yielding the Gini Importance score for each feature. These scores provide a quantitative measure of feature relevance, facilitating automatic feature selection based on their importance rankings.

Random Forest's Gini Importance offers several advantages, including interpretability, efficiency, and robustness [111]. Quantifying the impact of features on predictive performance enhances model interpretability and aids in understanding which features contribute most significantly to classification accuracy. Furthermore, its averaging across multiple decision trees improves efficiency and reduces the risk of overfitting, making it robust to noisy data and outliers. Feature selection using Random forest falls under the category of Embedded methods, which combine the qualities of filter and wrapper methods [103].

3.2.2 *L1-norm in Logistic Regression*

In logistic regression, the L1 norm, also known as Lasso regularization, is utilized for feature selection by penalizing the absolute values of the coefficients associated with each feature. The L1 norm adds a regularization term to the logistic regression cost function, which is the sum of the error between predicted and actual values, along with the absolute values of the coefficients multiplied by a regularization parameter (λ). This regularization encourages sparsity in the coefficient values, effectively driving some coefficients to zero, thus performing automatic feature selection [149].

The cost function of logistic regression with L1 regularization can be expressed as Equation 4.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right) + \lambda\sum_{j=1}^{n}|\theta_j| \quad (4)$$

where θ represents the coefficients (including the intercept term), x denotes the feature matrix, y is the label variable, $h_\theta(x)$ is the logistic function, m is the number of samples, n is the number of features, and λ is the regularization parameter. The term $\sum_{j=1}^{n}|\theta_j|$ denotes the L1 norm of the coefficients. It is particularly useful when dealing with high-dimensional datasets with many irrelevant or redundant features, as it helps in identifying and focusing on the most important features while ignoring noise or less relevant ones [149].

3.2.3 *L1-norm in Support Vector Machines*

The L1 norm in Support Vector Machines (SVM) for feature selection operates by incorporating an L1 penalty term into the SVM's optimization objective. This approach encourages sparsity in the model parameters, effectively driving some feature weights to zero, thus performing feature selection [47]. In contrast to the traditional SVM that focuses on maximizing the margin between different classes while penalizing classification errors, the L1-regularized SVM adds an additional layer of complexity by also penalizing the absolute value of the weights. The L1-regularized SVM can be expressed as minimizing the objective function as in Equation 5.

$$\min\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \lambda\|w\|_1\right) \quad (5)$$

Table 2: Comparison of Random Forest Gini Importance, L1-norm Logistic Regression, and L1-norm Support Vector Machine from different aspects. The differences show that each of these feature selectors can be suitable for a specific dataset with different kinds of relationships between features.

Feature	**RF Gini Importance**	**L1-norm LR**	**L1-norm SVM**
Methodology	Reducing the impurity across trees	Pushing coefficients to zero using L1 penalty	Affecting feature weights using L1 penalty
Use Cases	Complex dataset with non-linear relationships, robust to overfitting, handles categorical features	High-dimensional datasets with linear relationships, interpretable	High-dimensional, sparse datasets, handles both linear and non-linear separations
Advantages	Non-parametric, less sensitive to outliers, models complex interactions.	Straightforward interpretation, valuable for explanatory purposes and prediction.	Handles linear/non-linear boundaries, robust in high-dimensional spaces.
Limitations	Biased towards features with more categories; doesn't eliminate but ranks features.	Assumes a linear relationship between features and targets, which may not always hold.	Requires careful tuning of parameters; computationally intensive for large datasets.

where, $\|w\|_1$ is the L1 norm of the weight vector w, ξ_i, are the slack variables representing misclassification errors, C is a parameter that controls the trade-off between maximizing the margin and minimizing the misclassification error, and λ controls the strength of the L1 penalty encouraging sparsity in the weight vector.

L1-regularized SVMs are particularly effective for datasets where feature selection is crucial, typically in high-dimensional spaces with a large number of features but a relatively small number of samples. This scenario is useful for domains where the data might be sparse, and the relevant features for making accurate predictions are a small subset of the total features available.

However, the effectiveness of L1-regularized SVMs can depend on the choice of the regularization parameter λ, which requires careful tuning [10]. Moreover, this approach works better when there is an assumption that the impact of the majority of features on the target variable is minimal or that the data contains many irrelevant features. In contexts where most features contribute equally to the decision boundary or in very low-dimensional datasets, the benefits of L1 regularization might not be as pronounced.

3.3 COMPARISON BETWEEN DIFFERENT FEATURE SELECTION APPROACHES

Table 2 presents a comparison of the aforementioned feature selection methods. As highlighted, each method offers distinct advantages that are beneficial for different types of datasets. Consequently, employing a combination of these approaches can enhance the quality of the final feature subset, potentially making the methodology more versatile and applicable to a broader range of datasets.

3.4 PROPOSED PREPROCESSING PIPELINE

The exploratory data analysis in Section 3.1 reveals that network traffic datasets exhibit considerable differences. Supervised ML models discern different categories by learning the relationship between the feature space and ground-truth labels. In order to improve the generalization of supervised learning models, one effective approach is to augment the training data by incorporating additional network traffic datasets. To ensure the adaptability of ML-based NIDS to new network traffic data, a standardized preprocessing pipeline is essential. The proposed preprocessing pipeline is demonstrated in the first stage of Figure 7.

According to the EDA findings, certain features exhibit high linear correlations, implying redundancy in the information they provide to the ML model. Additionally, some features display zero variance, rendering them non-informative for the ML model. Therefore, features exhibiting a correlation exceeding 95%, as per Pearson correlation analysis, are pruned. Furthermore, features with zero variance are removed, as they do not contribute to distinguishing between attack and benign flows. In addition, certain flow features may introduce biases in the model, leading it to make decisions relying solely on this subset of information, thus limiting its ability to generalize across different flow samples with different distributions. These features include five-tuple features, which have a direct relation to the ground-truth labels (Benign/Attack), and time-related features, which can be different based on each network topology, network management rules, and the available concept drifts. These features are also pruned in our preprocessing pipeline. Therefore, in the first stage (statistical approaches), we effectively reduced the number of features to 45. To summarize, the preprocessing pipeline is presented in Algorithm 1.

Algorithm 1 : Preprocessing pipeline to create an appropriate flow features for training supervised ML-based NIDSs

```
Input : Network traffic dataset converted with NFStream tool to the feature
        space of $F = \{f_1, f_2, ..., f_k\}$ with the mean of features $\mu = \{\mu_1, \mu 2, ..., \mu_k\}$
Output : Reduced feature set
1 Preprocessing:
2 $Five\ tuple\ features\ \&\ time-related\ features \leftarrow Remove;$
3 if $\frac{1}{n}\sum_{i=1}^{n}(f_{ji} - \mu_j)^2 = 0$ then
4     $f_j \leftarrow Remove;$
5 end
6 if $|\rho(f_i, f_j)| > 0.95$ then
7     $f_i \leftarrow Remove;$
8 end
```

3.5 PROPOSED ENSEMBLE FEATURE SELECTION

After reducing the size of flow features through the preprocessing pipeline (Section 3.4), we implement an Ensemble Feature Selection (EFS) method, depicted in the second stage of Figure 7. This process aims to decrease the dimensionality of flow features, which can impact the complexity of the supervised ML model and its ability to generalize detection performance. This becomes crucial if retraining the ML model used in the NIDS is necessary, such as in online learning approaches, or if computational resources are limited [60].

However, it is vital to ensure an accurate selection of informative features for imbalanced network traffic datasets. This entails retaining not only features crucial for classes with a majority of samples but also preserving information relevant to all categories. Additionally, network traffic datasets show different patterns due to extracting from different network architectures with various network management rules and routing protocols [39]. This inherent diversity results in variations between network traffic datasets. Consequently, these differences pose challenges for feature selection approaches, especially when focusing solely on one network traffic dataset. Features selected utilizing one network traffic dataset may extract important characteristics of network traffic patterns with only similar behavior.

As discussed in Section 3.3, each feature selection method has its own strengths and limitations. To design a feature selection method capable of being applied to various network traffic datasets and detecting flow features transferable to others, we combine their output results with certain considerations in this method.

Within the proposed EFS, each network traffic dataset undergoes three distinct approaches: Lasso Logistic Regression (LR), Lasso Support Vector Machine (SVM), and Random Forest (RF) Gini Importance. If a network traffic dataset encompasses multiple attack types, separate datasets are created for each attack and benign flows. This approach mitigates bias in differentiating between benign and attack flows, particularly in datasets where certain attack types dominate, potentially overshadowing crucial features.

The output of each feature selection method comprises coefficient values assigned to individual features, indicating their importance. Subsequently, features are ranked in descending order of coefficient values. The top features from each set are selected, followed by the intersection of these selections. This process yields the final feature set, comprising the 5, 10, 15, 20, and 30 most important flow features.

To improve the transferability of our proposed EFS method, we introduce a data-driven approach known as DD-EFS [59]. In this approach, the EFS method is employed across multiple network traffic datasets to extract the top 20 features from each. This selection (top 20 features) is based on the optimal performance achieved by the EFS method. Subsequently, an intersection is taken over these features, resulting in 5 common network traffic flow features. By utilizing multiple network traffic datasets, which differ according to EDA results (mentioned in Section 3.1), to select these 5 features, the likelihood of their transferability to new, previously unseen network traffic datasets is increased.

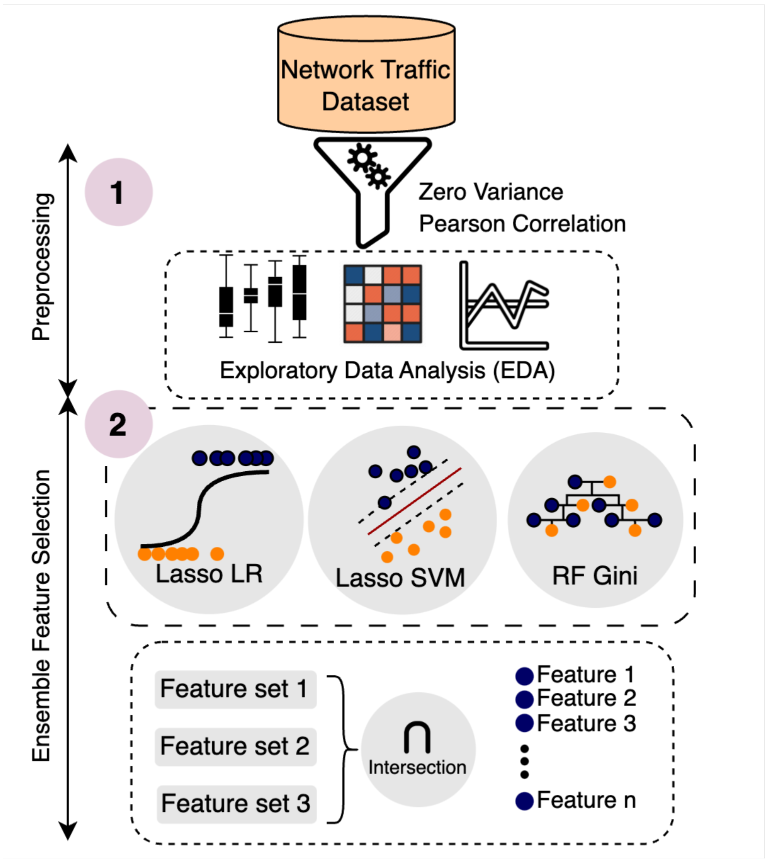

Figure 7: The general architecture of the proposed preprocessing pipeline and the proposed Ensemble Feature Selection (EFS) approach. The first stage illustrates the preprocessing pipeline crafted following exploratory analysis of network traffic datasets. Subsequently, the second stage demonstrates the EFS approach, leveraging the capabilities of three ML-based feature selectors to enhance the method's robustness.

3.6 EVALUATION DESIGN

In this section, we elaborate on our evaluation design to explore our proposed EFS from different perspectives. We utilize multiple supervised ML models to assess the efficacy of our proposed EFS approach [57]. To evaluate both the robustness of the proposed method and the transferability of the selected features, we utilize five network traffic datasets. These network traffic datasets were chosen for their differences, as revealed in the EDA discussed in Section 3.1, and their availability in pcap format. More details about the datasets can be found in Section A.2.

In this section, we first investigate the influence of the number of selected relevant features on detection performance and its impact on the training and evaluation time of ML models. Furthermore, to assess the effectiveness of designing an ensemble method, we evaluate detection performance using features selected by each individual feature selection method and compare it with the scenario where they are combined into an ensemble (EFS). Additionally, to understand the transferability of selected features across various network traffic datasets with diverse distributions and attack types, we augment the proposed EFS approach by applying it to multiple network traffic datasets. The transferability of these selected features is investigated in comparison to extracting features solely from one network traffic dataset.

3.6.1 *Datasets*

This section provides details about the datasets utilized for both training and evaluation purposes: CICIDS17, CTU13, UNSW-NB, Botnet, and SlowDoS. Each network traffic dataset, as explained in Section A.2, is divided into training, validation, and test datasets, with proportions of 70%, 10%, and 20% of the total dataset, respectively.

To optimize each ML model's detection performance, we employ cross-validation techniques using the training and validation sets. The test set remains unseen until the final evaluation stage. Consequently, the models' hyperparameters are chosen based on the results of cross-validation conducted on the validation dataset.

To assess the DD-EFS approach, results are extracted using CICIDS17, CTU13, and UNSW-NB datasets, and their transferability is investigated on two different, previously unseen datasets: Botnet and SlowDoS.

3.6.2 *ML Models Used in Evaluation Scenarios*

To evaluate the effectiveness of our proposed preprocessing techniques, EFS algorithm, and DD-EFS algorithm, we conducted experiments using a range of supervised ML models. These models include:

- Random Forest (RF) model, comprising 200 decision trees.
- Logistic Regression (LR) model, with a maximum of 200 iterations.
- Multi-Layer Perceptron (MLP) model, consisting of four layers with neuron counts of 20, 32, 20, and 1, respectively. Batch normalization is incorporated into each layer, and dropout regularization with a rate of 0.3 is applied to the second and third layers. Binary cross-entropy is employed as the loss function, while the Adam optimizer is used for parameter optimization.

Further details regarding the model structure can be found in Section 2.4.1.

3.6.3 *Evaluation Metrics*

To evaluate the efficacy of the proposed EFS method, it is crucial to assess the detection performance of the ML model. For this purpose, we utilize the macro-average of the F1-score metric. The F1-score is chosen as it accounts for both false positives (FP) and false negatives (FN), providing a comprehensive evaluation of the detection performance.

In this study, the positive class denotes attacks, while the negative class represents benign flows. Recognizing that benign flows typically constitute the majority of network traffic, we opt to compute the macro-average of the F1-score. This approach ensures equal consideration of each class, irrespective of its frequency or potential imbalance in the dataset [57].

To compute the F1-score, precision and recall are derived using Equations 6 and 7, respectively. Subsequently, the F1-score and its macro-average are calculated based on Equations 8 and 9.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

Here, TP refers to True Positive, representing the number of attack samples that are correctly identified as attack samples. FPs represent benign samples that are incorrectly detected as attack samples.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

Where FNs represent attack samples that are incorrectly detected as benign samples.

$$\text{F1-score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{8}$$

$$\text{Macro-averageF1-score} = \frac{1}{N} \sum_{i=1}^{N} F1_i \tag{9}$$

Where N denotes the number of classes, with two classes representing "Benign" and "Attack" samples.

3.6.4 *System Information*

To implement the proposed preprocessing pipeline, EFS, DD-EFS, and train ML models, we utilize a Ubuntu server equipped with 128GB RAM and 4 CPUs (Intel Core i9-10900K). Moreover, the implementation is done in Python, using the Scikit-learn, pandas, and PyTorch libraries. Each evaluation is independently run five times to present the mean value of the results, ensuring the robustness of the final values.

3.7 EVALUATION RESULTS

In this section, we conduct experiments to investigate the scenarios discussed in Section 3.6.

3.7.1 *Selected Flow Features*

This section presents the final coefficient values of each flow feature extracted using the proposed EFS method. To address variations in the outputs of Lasso LR, Lasso

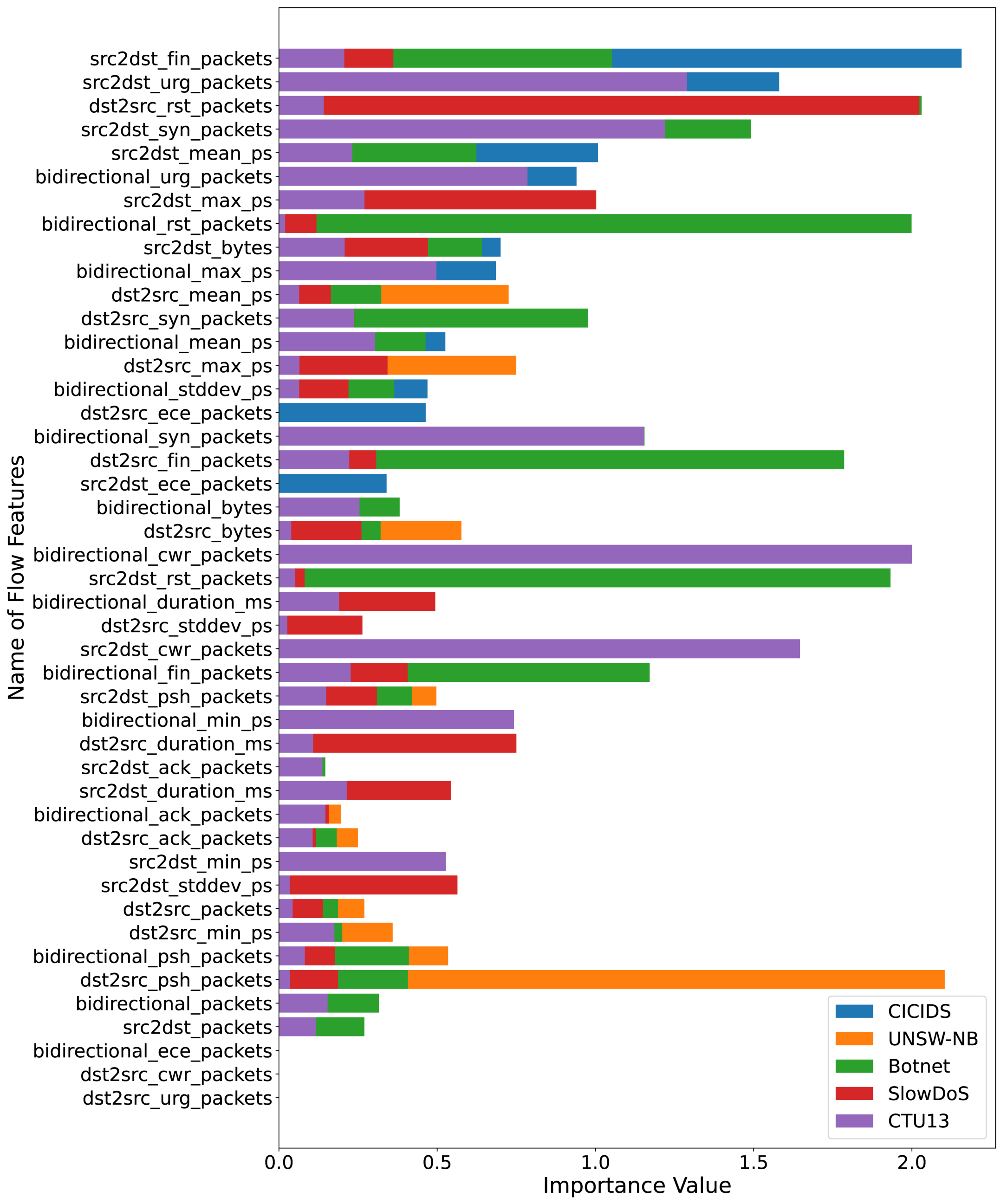

Figure 8: Feature importance values obtained through the proposed EFS method for five various network traffic datasets. The coefficient value of each individual feature selection method is normalized between 0 and 1, so the maximum value that a feature importance can have is 3. *This figure is extracted from [59].*

SVM, and RF feature selection methods, we normalize their outputs to ensure they fall within the range of 0 to 1. This normalization prevents any individual selector from disproportionately influencing the final coefficient values, ensuring that the total sum of all coefficients does not exceed three. Analyzing the coefficient values for each feature across different network traffic datasets, as illustrated in Figure 8, highlights variations in feature relationships among the datasets. This observation indicates that different network traffic datasets exhibit distinct flow patterns due to their unique attack types and characteristics. Consequently, employing a feature selection algorithm on a single dataset and applying the selected features across diverse datasets to construct a generalized ML-based NIDS is ineffective.

Hence, it becomes essential to explore which features are transferable among various network traffic datasets. This challenge is addressed through the design of DD-EFS.

3.7.2 *Impact of Feature Selection on Detection Performance*

In this section, we evaluate the detection performance of multiple ML models using different feature sets, including the 5, 10, 15, 20, and 30 most relevant features. We compare the detection performance achieved by incorporating feature importance with a scenario where only a preprocessing pipeline (reducing feature size from 88 features to 45 features) is applied to the network traffic feature files, utilizing all 45 features during the training process.

Based on the findings presented in Table 3, it is observed that the performance achieved without the proposed EFS using all 45 features can be achieved by using only 15 or 20 features. This indicates a potential reduction in the feature set by 67% or 56%, respectively, while still maintaining comparable detection performance to that obtained with the entire feature set.

According to Table 3, the RF model consistently demonstrates high detection performance across various network traffic datasets. Specifically, when trained on 20 selected features, its detection performance closely aligns with scenarios where no feature selection is applied. Remarkably, expanding the feature set from 5 to 15 features notably enhances detection performance in the CICDoS and Botnet datasets while yielding minor improvements in other network traffic datasets. The MLP model exhibits similar detection performance to the RF model across most datasets, except for the Botnet dataset, where it achieves comparatively lower detection rates.

Both the RF and MLP models showcase the potential to maintain high detection performance even with reduced feature dimensions, similar to scenarios without feature selection. Conversely, for the LR model, reducing the feature dimension to smaller selected subsets can also enhance detection performance in specific network traffic datasets such as CICIDS17, CICDoS, and UNSW-NB.

3.7.3 *Impact of Feature Selection on Training Time*

One primary objective of selecting the most relevant features is to reduce feature dimensions, thereby minimizing noise within features and speeding up the training time of ML models. This becomes particularly crucial when employing ML-based NIDS in online scenarios [60]. In scenarios requiring retraining of the ML model, it's crucial to ensure that updating ML-based NIDS doesn't introduce significant time

Table 3: Comparison of the detection performance of RF, LR, and MLP-based NIDS across diverse network traffic datasets, with and without the proposed EFS. It highlights the impact of feature size on detection performance, emphasizing the trade-off between information loss and detection effectiveness. It also shows the efficiency of EFS in optimizing detection performance without utilizing all available features.

Dataset	F1-score (%)					
	Without EFS	5 Features	10 Features	15 Features	20 Features	30 Features
CICIDS17:						
RF	99.8	98.1	99.4	99.4	99.5	99.4
LR	74.3	94.3	99.0	94.1	89.1	73.2
MLP	99.9	99.2	99.2	99.0	99.6	99.6
CICDoS:						
RF	99.1	80.2	94.6	99.0	99.0	99.0
LR	82.3	75.1	86.4	86.2	90.2	82.5
MLP	96.0	80.0	93.1	94.4	95.0	95.0
CTU-13:						
RF	99.9	95.9	97.3	99.1	99.4	99.4
LR	93.0	94.2	94.2	93.0	94.1	90.0
MLP	98.7	94.3	94.4	97.2	97.2	97.7
Botnet:						
RF	94.6	68.3	85.1	93.2	94.4	94.5
LR	64.2	57.8	60.0	62.3	63.1	64.0
MLP	79.7	70.1	80.0	80.1	80.0	80.2
UNSW-NB:						
RF	99.9	99.0	99.9	99.9	99.9	99.9
LR	74.7	94.2	99.0	94.2	89.5	73.8
MLP	99.9	99.1	99.9	99.9	99.9	99.9

overhead. This is essential to avoid increased network latency, potential packet loss, and a decline in Quality of Service (QoS).

In this section, we investigate the impact of reducing feature dimensions on the training and evaluation times of each supervised ML model. Table 4 showcases the results for one large (i.e., CTU-13) and one small (i.e., Botnet) network traffic dataset. Additional results for other datasets can be found in Section A.6. Our analysis in Table 4 reveals that RF and MLP models exhibit longer training times, mainly owing to their structural complexity, unlike LR, which is a linear model. However, the results show that the training time of RF is highly influenced by the number of features, with an increase in feature size leading to a corresponding increase in training

Table 4: The training time required for three distinct ML models, including RF, LR, and MLP, across varying feature dimension sizes for both the largest and smallest network traffic datasets. As demonstrated, the training time is influenced by both the number of data points and the feature dimension size.

Dataset	**Training Time (Second)**			
	5 Features	15 Features	30 Features	45 Features
CTU-13 with 3,329,312 flow samples:				
RF	63.19	253.13	533.52	592.54
LR	4.53	24.35	30.12	47.78
MLP	274.41	264.56	268.07	276.91
Botnet with 182,999 flow samples:				
RF	3.62	17.87	36.59	33.89
LR	0.16	0.91	0.79	2.55
MLP	14.57	15.45	17.89	19.23

time. Conversely, MLP training time shows only a slight increase, suggesting it is less dependent on feature size. Additionally, a comparison across two distinct datasets reveals that retraining time is influenced not only by the feature size but also by the dataset size. Specifically, in datasets with a greater number of flow samples, training time tends to increase.

3.7.4 *Effectiveness of Ensemble Flow Feature Selection*

In this section, we compare detection performance when the models are trained on feature sets received from individual feature selection with the case where features are extracted from the proposed EFS approach. This comparison can show the effectiveness of combining the feature selection methods over using a single feature selection approach.

Selecting Features Leveraging Solely RF Gini Importance

In the following analysis, feature selection is conducted solely based on the RF Gini importance method. The ML models are then trained using subsets comprising 11% (5 most important features), 22% (10 features), 33% (15 features), 44% (20 features), and 66% (30 features) of the total features. The results are presented in Table 5. A comparison between Table 5 and Table 3 reveals that features selected through RF Gini importance significantly impact the detection performance of the

Table 5: Detection performance of different supervised ML-based NIDS across various network traffic datasets with selected features from solely RF gini importance selector.

Dataset	**F1-Score (%)**				
	5 Features	10 Features	15 Features	20 Features	30 Features
CICIDS17:					
RF	96.0	96.2	96.3	96.7	97.0
LR	91.7	92.2	92.1	89.1	72.9
MLP	91.1	96.2	94.4	94.5	96.1
CICDoS:					
RF	98.0	99.1	99.2	99.1	99.1
LR	74.3	82.9	83.2	85.8	78.1
MLP	78.1	93.2	94.4	94.5	94.7
CTU-13:					
RF	95.4	98.3	99.1	99.3	99.3
LR	86.3	87.1	87.4	81.2	80.0
MLP	91.0	93.6	95.2	97.0	97.1
Botnet:					
RF	93.4	94.2	94.3	94.3	94.3
LR	56.1	60.0	60.1	60.1	62.2
MLP	70.0	74.2	78.5	74.3	77.1
UNSW-NB:					
RF	99.2	99.7	99.9	99.9	99.9
LR	83.1	86.9	84.8	83.2	82.1
MLP	98.2	99.2	99.2	99.3	99.3

RF model. This is likely due to the compatibility between the model's architecture and the feature selection method, leading to improved detection performance. However, utilizing features selected solely via the RF Gini importance method results in decreased detection performance for the MLP and LR models compared to when the EFS method is employed.

The outcomes obtained from selecting features solely based on L1-norm LR and L1-norm SVM are presented in Section A.7. A comparison between the detection performances achieved when either of these models is used individually and when they are combined through the EFS method reveals that ensembling leads to improved detection performance, even with a smaller subset of flow features.

3.7.5 *Transferability of Selected Flow Features*

In this section, to select a transferable feature set, we introduce a data-driven solution and integrate it into the proposed EFS method, naming it DD-EFS. Considering the results of Exploratory Data Analysis (EDA) and the coefficient values of each network traffic dataset, it is evident that these datasets can differ from one another. Moreover, in real-world scenarios, the emergence of new, previously unknown attacks and concept drift in network traffic distribution are common. Therefore, the selected features should demonstrate high detection performance not only on similar pattern flows but also on different patterns.

In DD-EFS, the selected features consist of the intersection of the top 25 features from three datasets: CICIDS17, CTU13, and UNSW-NB. This approach ensures that the selected features are not based solely on one dataset but are instead derived from multiple datasets, enhancing their potential transferability and robustness. The selected 5 common features include "bidirectional maximum packet size", "bidirectional mean packet size", "source to destination maximum packet size", "destination to source SYN packets", and "bidirectional duration in milliseconds" [59].

To assess their transferability, the ML models are trained on these 5 features of Botnet and SlowDoS network traffic datasets, which were excluded during the selection process. The detection performance is shown in Table 6. The performance of

Table 6: Investigating transferability of the selected features from *DD-EFS*. Here, DS1, DS2, and DS3 refer to CICIDS17, UNSW-NB, and CTU13 datasets, respectively. *The table is extracted from [59].*

Dataset	**F1-Score (%)**			
	5 top features of DS1	5 top features of DS2	5 top features of DS3	*DD-EFS* features
SlowDoS				
RF	96.3	98.1	76.8	**99.0**
LR	71.6	64.3	71.2	**72.7**
MLP	91.1	95.2	76.4	**95.5**
Botnet				
RF	85.9	82.7	70.4	**91.8**
LR	60.3	59.3	60.8	**64.8**
MLP	79.2	70.6	70.4	**83.2**

DD-EFS is compared to the scenario where the features are selected solely from one of the datasets using EFS. According to the results, DD-EFS achieves higher detection performance on both previously unseen network traffic datasets compared to selecting features solely from one network traffic dataset. This highlights the effectiveness of DD-EFS in identifying transferable features across diverse datasets. The highest detection performance is shown in bold.

Since the features extracted from DD-EFS are not the top 5 features of each network traffic dataset individually, we also evaluate the detection performance on each dataset and present the results in Table 7.

According to the results, while the 5 features extracted from the corresponding network traffic dataset can achieve the highest detection performance, the features extracted from DD-EFS achieve the second-best detection performance with only a slight difference. This suggests that while dataset-specific features may offer slightly better performance on their respective datasets, the transferable features selected by DD-EFS still demonstrate competitive detection performance across different datasets. Therefore, we can infer that these selected features not only exhibit high performance but also encapsulate crucial information necessary for detecting various intrusions across diverse network traffic scenarios.

Table 7: Investigating Detection performance of the selected features from *DD-EFS* for network traffic datasets that were involved in *DD-EFS*. Here, DS1, DS2, and DS3 refer to CICIDS17, UNSW-NB, and CTU13 datasets, respectively.

Dataset	**F1-Score (%)**			
	5 top features of DS1	5 top features of DS2	5 top features of DS3	*DD-EFS* features
CICIDS17				
RF	98.1	95.0	88.2	96.1
LR	94.3	89.2	88.1	90.3
MLP	99.2	93.1	87.9	94.6
UNSW-NB				
RF	98.1	99.0	50.0	98.8
LR	56.7	94.2	50.0	73.8
MLP	69.4	99.1	50.0	92.8
CTU13				
RF	90.2	90.0	95.9	95.0
LR	81.9	55.3	94.2	87.4
MLP	90.8	86.8	94.3	92.9

3.8 SUMMARY

In this chapter, we conduct an exploratory data analysis (EDA) to illustrate the diversity of network traffic datasets. This analysis highlights that using only one method to extract the most relevant features may work well for one dataset but may not be suitable for others.

Therefore, we introduce a new ensemble feature selection (EFS) method [57] that leverages statistical and ML-based feature selection techniques to reduce feature dimensionality while maintaining robust detection performance. By employing our proposed EFS method, a reduction in feature dimension by 56% is achievable while maintaining detection performance comparable to using all features. Additionally, we demonstrate that utilizing the EFS method leads to decreased training time, which is crucial when updating the ML model is necessary [60].

Lastly, we explore the possibility of extracting transferable flow features across diverse network traffic datasets by providing a data-driven solution and integrating it into the proposed EFS method (DD-EFS) [59]. The results show that features extracted from DD-EFS can be transferable across different, previously unseen network traffic datasets compared to features selected solely from one network traffic dataset.

4 SELF-SUPERVISED NETWORK INTRUSION DETECTION SYSTEM

In this chapter, Sections 4.6.2, 4.8.2, 4.8.3, and 4.8.7 are taken verbatim from [58].

The performance of a Machine Learning (ML)-based Network Intrusion Detection System (NIDS) can vary depending on various factors such as the composition of its training dataset, the structure of its ML model, and the distribution of its evaluation dataset. For instance, supervised ML-based NIDSs can achieve high detection performance for network flows within a dataset (intra-dataset) through a proper preprocessing and feature selection approach [57]. However, studies indicate that their performance degrades significantly when evaluated with traffic patterns different from the training dataset, a phenomenon known as cross-domain detection performance or inter-dataset generalization of an ML-based NIDS [3, 96, 179].

Definition — *Cross-Domain Detection Performance* denotes the efficacy of an ML-based NIDS when trained on one dataset and tested on another dataset that may have a distinct distribution [39, 94].

This is an important issue to consider, as the traffic patterns within a network are influenced by various factors such as network architecture, deployed network services, network security management rules, and the number of employees accessing the network [179]. The variability in the network environment can directly impact the behavior of network users and network traffic flow features.

Therefore, the ML-based NIDS trained on network traffic collected from a specific network setting may exhibit high detection performance when analyzing flows originating from the same network pattern. However, its detection performance may decline when dealing with new flows from a different network environment. This mirrors real-world situations where the NIDS encounters diverse network traffic patterns due to the heterogeneous nature of network data [8]. Additionally, the emergence of new network intrusions may exhibit distributions differing from those present in the training dataset, and detecting them accurately is necessary.

Furthermore, supervised learning models primarily learn the relationship between flow features and ground truth labels, thereby lacking the ability to discern abstract representations of the underlying nature of a flow. Consequently, they encounter difficulties in detecting new network intrusions that may mimic benign flow patterns, such as botnets and multi-stage attacks (MSA) [12]. Additionally, collecting and accurately annotating such intrusions can pose difficulties for humans, thereby limiting the availability of various annotated network intrusion flows for training supervised ML models.

Conversely, unsupervised ML models do not need annotated datasets and can acquire more abstract representations of features. However, it primarily learns dataset-specific features through feature reconstruction [167]. While this approach can be advantageous in discovering underlying patterns in unlabeled data, it also poses challenges when faced with network flows divergent from the training data pattern. In such cases, the model's ability to generalize may be limited, leading to decreased detection performance akin to the constraints observed in supervised learning.

Therefore, both supervised and unsupervised models encounter challenges in achieving high cross-domain detection performance, a crucial metric for designing a *generalized ML-based NIDS*. This chapter introduces a novel approach using self-supervised learning to address these challenges by learning abstract representations of only benign flows.

4.1 PROBLEM STATEMENT

Training an ML-based NIDS on a limited set of network traffic flows can diminish the model's ability to generalize to other datasets with different distributions, which is referred to as cross-domain detection performance.

To illustrate the issue, suppose the ML-based NIDS is trained on a training dataset $D_s = \{X_s, P_s(x), x \in X_s\}$, where X_s represents the feature space and $P_s(x)$ denotes the marginal probability distribution of all samples in the feature space. In scenarios where network traffic exhibits heterogeneity due to diverse user behaviors, network management rules, and varying network topologies, it is highly probable to encounter another network traffic dataset $D_t = \{X_t, P_t(x), x \in X_t\}$ with probability distributions $P_s(x) \neq P_t(x)$. Even if the ML-based NIDS f is trained on D_s using a specific learning algorithm A to minimize a loss function L, its detection performance on D_t may not match the performance achieved on D_s. In fact, its performance may deteriorate significantly on D_t. Thus, $\mathrm{Performance}(f, D_s) \neq \mathrm{Performance}(f, D_t)$. This disparity in performance underscores the presence of the cross-dataset problem, where the model's effectiveness on the training dataset does not perform well in the testing dataset.

To address this problem and align the ML-based NIDS more closely with real-world scenarios, where access to all network traffic and its varied distributions, as well as the emergence of new attacks, is not feasible, we need to design a model that doesn't rely on annotated attack flows. This is because capturing the traffic and pattern of a novel attack can be time-consuming, and depending solely on annotated data may hinder the ML-based NIDS's ability to detect previously unseen threats effectively. Therefore, this model should be capable of learning abstract representations of network traffic to enable effective discrimination between benign and malicious entities.

4.2 PRETEXT TASKS

The Self-Supervised Learning (SSL) method, as defined in 2.4.3, is categorized under unsupervised learning, eliminating the requirement for annotated network traffic data. Unlike conventional unsupervised learning models, SSL can discern more abstract representations of data by optimizing pretext tasks and harnessing inherent properties and content within the dataset [31, 156]. In the following, some well-known pretext tasks utilized in SSL are succinctly elaborated.

Pretext tasks serve as mechanisms to enable ML models to grasp the underlying structure or relationships within the data [161]. By training the model on these pretext tasks, it can be refined for downstream tasks, especially in situations where labeled data is scarce or costly [121].

In the domain of Natural Language Processing (NLP), common pretext tasks include word masking and word prediction within sentences [108]. Similarly, in computer vision, pretext tasks such as image colorization and image rotation prediction enable the model to learn invariant features and abstract representations of images, which can then be utilized for classification tasks [120]. For tabular data, adding noise to features through column or row scrambling or column or row masking and accurately predicting these modifications is regarded as a pretext task [163].

4.3 CONTRASTIVE LEARNING

Contrastive learning is a technique used in SSL to train models without labeled data. It operates by contrasting pairs of similar and dissimilar samples in the latent space [108]. Specifically, the model is trained to map similar instances (positive pairs) closer together while pushing dissimilar instances (negative pairs) farther apart. This is typically accomplished by employing contrastive loss functions, such as InfoNCE (Noise Contrastive Estimation) [129] or Normalized Temperature-Scaled Cross-Entropy (NT-Xent) loss [32], to measure the agreement between positive pairs and the divergence between positive and negative pairs. By optimizing this objective, the model learns to capture meaningful representations of the data that capture underlying structures or relationships, which can then be transferred to downstream tasks. Contrastive learning is particularly effective for SSL as it leverages the inherent structure of the data itself, making it well-suited for scenarios where labeled data is scarce or unavailable [109].

Creating similar and dissimilar instances, referred to as positive and negative pairs, respectively, involves the generation of variations or manipulations to the data, which is akin to data augmentation. Data augmentation techniques are widely utilized in machine learning to expand the training dataset artificially by introducing variations or transformations to the existing data. These techniques encompass various methods such as image transformations (e.g., cropping, rotation, flipping) [120], text transformations (e.g., shuffling words, paraphrasing) [108], and manipulation of tabular data (e.g., adding noise, shuffling rows or columns) [163]. These augmenta-

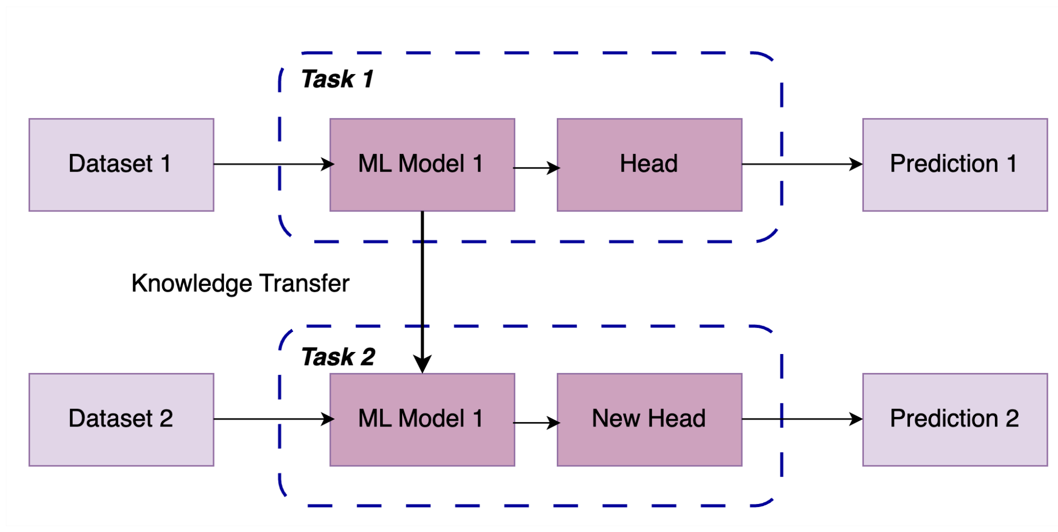

Figure 9: The idea of transfer learning, where one model is trained for a specific task can be used as a pre-trained model for solving another task.

tions play a vital role in fostering the creation of diverse pairs and make the model's capacity to generalize to unseen data.

4.4 TRANSFER LEARNING

Transfer learning is a machine learning technique that utilizes knowledge from one domain (source) or task to another (target) domain. It is mostly used when the target task has a limited amount of annotated data available [58]. Transfer learning is widely utilized in computer vision, and natural language processing, where a machine learning model pre-trained on a source dataset is employed for prediction tasks on a target dataset [77]. It is important to note that pretraining can occur with or without label supervision from the source dataset [58]. In this chapter, we show that our proposed contrastive SSL NIDS can be used as the pre-trained model.

4.5 CONTRIBUTIONS

As detailed in Section 4.1, supervised ML-based NIDSs heavily rely on annotated network datasets to establish relationships between trained network traffic flow features and ground-truth labels. Conversely, unsupervised ML-based NIDSs do not require annotated data for training; however, they learn representations by reconstructing the training dataset, thus making their learned representations dependent on the specifics of the training data. Consequently, both supervised and unsupervised approaches face limitations in achieving high cross-domain detection performance.

The SSL methods discussed in Section 2.6.1 are categorized based on their utilization of label information in their pretext task and the generation of positive and negative pairs (explained in Section 4.3). Some methods use labels for this purpose,

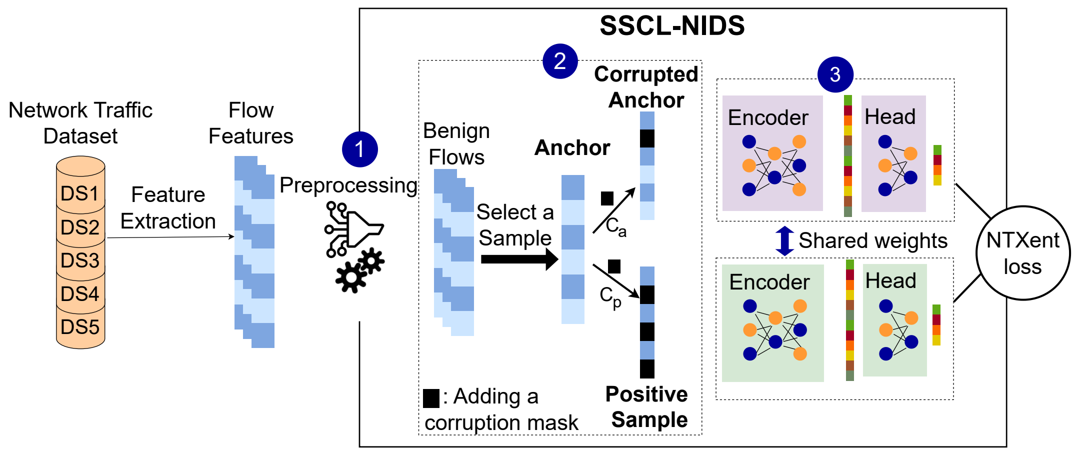

Figure 10: The general architecture of the proposed *SSCL-NIDS*. It comprises three main modules, including (1) preprocessing pipeline, (2) data augmentation, and (3) model training. *The figure is extracted from [58].*

while others remain unsupervised, devoid of label information. In our endeavor to reduce the dependency on annotated data, our proposed approach keeps the pretext task unsupervised. Furthermore, to tackle the generalization challenge in ML-based NIDSs effectively, we aim to design a model capable of learning generic representations of benign traffic patterns. Unlike other SSL-based NIDSs, we enhance the augmentation process by applying corruption masks to create both positive pairs and modified anchors. Additionally, we train the ML-based NIDS on benign flows from multiple network traffic datasets. This dual augmentation strategy, along with the inclusion of diverse network traffic patterns, enables the model to learn more generic representations of benign flows. Furthermore, we leverage our proposed model as a pre-trained model and explore its effectiveness in transfer learning scenarios. This approach allows us to assess the model's adaptability.

4.6 PROPOSED SELF-SUPERVISED CONTRASTIVE LEARNING

In this section, we introduce our approach aimed at increasing the generalization and cross-domain detection performance of ML-based NIDS, called Self-Supervised Contrastive Learning-base Network Intrusion Detection System (SSCL-NIDS) [58]. As depicted in Figure 10, which provides a general overview of the proposed SSCL-NIDS, this ML-based NIDS comprises multiple modules. These include a preprocessing pipeline, data augmentation, and model training, which are explained in the following sections.

4.6.1 *Flow Scope & Preprocessing Pipeline*

In this work, each flow comprises packets characterized by identical 5-tuple features, encompassing source and destination IP addresses, source and destination MAC addresses, and port numbers. As detailed in Section 2.3, we employ the flow aggregation tool NFStream [11] to aggregate network traffic packets into flow-based datasets and extract flow-based features. These extracted features are then categorized into specific groups, which are detailed in Section A.3.

To elucidate the flow feature extraction process employed in the training and evaluation of SSCL-NIDS, we provide an explanation of the NFStream implementation below. NFStream provides a flexible attribute configuration for flow feature extraction. The enabled attributes in our implementation, along with their purposes, are listed below:

- The "accounting_mode" is set to 3, allowing the accumulation of payload lengths for bytes-related features.
- "statistical_analysis" is enabled, facilitating the extraction of post-mortem flow statistical features.
- The "activation timeout" ("active_timeout" property) is set to 30 minutes, signifying that packets with similar 5-tuple features received within this timeframe are added to the indexed flow; otherwise, they are considered as a new flow.

Further processing of the aggregated flow-based dataset is required, as it still encompasses features of entire flows and early statistical features are stored as lists. We utilize Pandas (Python library) to eliminate redundant features, splitting the early statistical features into distinct values for each packet, before storing the complete flow-based dataset in a CSV file. This process is applied consistently across all five datasets to ensure uniformity in the number of flow-based features, totaling 88. The list of features can be found in Section A.1.

Ground-truth labels (Benign/Attack) are assigned based on information about victim hosts and attacker IP addresses provided within each dataset, which are used solely for evaluating the proposed SSCL-NIDS.

The preprocessing pipeline, which is similar to the preprocessing pipeline in Section 3.4, involves removing certain features prone to leaking information and biasing ML models. Specifically, 5-tuple features exhibiting high linear correlations with ground-truth labels are removed, along with time-related features, to prevent information leakage of dataset-specific network topology and management rules. Only the "duration" feature, essential for certain attack types like DoS and DDoS, is retained, given its significance and lack of specificity across datasets and scenarios. Additionally, features with zero standard deviation are filtered out as they lack informativeness and may introduce noise. This preprocessing reduces the feature dimension to 45, which are available in Section A.3.

4.6.2 *Data Augmentation*

This section is taken verbatim from [58].

In SSL methods, augmentation generates additional training data by applying different transformations (here, we use a corruption mask) to the existing training data [31]. In SSCL-NIDS, to create a positive pair (i.e., semantically similar samples), an augmented view of an anchor (original sample) is generated by incorporating the content of the sample. Each augmented sample represents a variant of the anchor sample with subtle differences, preserving essential semantic information. This approach ensures the creation of meaningful pairs for effective contrastive learning. In this work, to generate a positive pair, we randomly apply a corruption mask with two different corruption rates of C_p and C_a to two subsets P and A of the anchor's features. These subsets are randomly selected from the original set of features $F = \{f_1, f_2, ..., f_M\}$, where $|P| = C_p \times M$ and $|A| = C_a \times M$ respectively. The value of the j_{th} corrupted feature $\hat{f_j}$ is selected uniformly from the empirical marginal distribution of f_j; hence $\hat{f_j} \sim \text{Uniform}(f_j)$. The marginal distribution of each feature is calculated for the entire dataset initially. The method of extracting the value of the corrupted feature from the marginal distribution aligns with the structure of traffic flow features, which often includes diverse numerical scales and types. Integrating the positive corruption rate provides control over the dissimilarity among positive pairs. Larger values can alter all feature values, whereas smaller corruption rates only affect a limited subset of features. This results in a more straightforward optimization task and a less robust learned representation. The algorithm of corruption mask and making augmentation is also summarized in Algorithm 2.

4.6.3 *Model Architecture*

In the proposed SSCL-NIDS method, as illustrated in Figure 10, a dual model architecture consisting of an encoder network, denoted as g, and a head network, denoted as h, are introduced. Each of them is explained in the following sections.

Encoder:

The encoder plays a crucial role in learning a compressed latent representation of the raw data. This representation captures significant features or attributes of the input data in a lower-dimensional space, aiming to extract features beneficial for downstream tasks without explicit supervision. In SSCL-NIDS, an MLP-based encoder g is employed, comprising five hidden layers with 45, 64, 128, 64, and 45 neurons, respectively, which are the same as related work. The encoder is trained using augmented data generated by a corruption mask, aiming to produce latent representations conducive to the contrastive learning task. It endeavors to bring similar augmented samples closer while pushing dissimilar ones farther apart in the latent space. The trained encoder can subsequently serve as a pre-trained model for transfer learning.

Head:

The head also referred to as the decoder or task-specific module, is responsible for executing specific tasks leveraging the learned representations from the encoder. The output of the encoder is fed into the head network h, which consists of a projection head followed by a normalization layer. This configuration prepares the features for computing the contrastive loss. Additionally, the weights of the encoder and projection head are shared to ensure consistent representation learning across diverse views of the same input, as depicted in Figure 10.

Contrastive Loss Function:

To minimize the distance between similar representations for positive pairs $(z_i, \hat{z_i})$ while simultaneously maximizing the distance between dissimilar representations for negative pairs (z_i, z_j), we employ the NTXent loss function [32].

$$\text{NTXent}(z_i, \hat{z}_i) = -\log\left(\frac{\exp\left(\text{sim}(z_i, \hat{z}_i)/\tau\right)}{\sum_{j=1}^{2N-1}\exp\left(\text{sim}(z_i, z_j)/\tau\right)}\right) \tag{10}$$

The NTXent loss is defined by Equation 10, where $\text{sim}(z_i, \hat{z}_i)$ represents the similarity score between positive pairs and $\text{sim}(z_i, z_j)$ denotes the similarity score between the anchor and other samples in the mini-batch. Moreover, τ serves as the temperature parameter, scaling the logits before applying the softmax activation function.

Additionally, the temperature parameter τ plays a crucial role in controlling the proximity of similar data points. A smaller value of τ results in a higher penalty, compelling the model to position semantically similar data points closer to each other. Conversely, a larger value of τ leads to a softer penalty, allowing for greater flexibility in the arrangement of similar representations within the latent space. This mechanism provides the model with the ability to adapt its clustering behavior based on the specific requirements of the task or dataset.

Algorithm 2 shows the proposed *SSCL-NIDS*, including its corruption mask procedure, the output of encoder and head models, and how to incorporate these elements into the loss function [58].

4.7 EVALUATION DESIGN

In this section, we design various evaluation scenarios and experiment setups to evaluate the detection and generalization performance of the proposed SSCL-NIDS using five distinct datasets as detailed in Appendix A.2. Generalization performance encompasses the model's ability to detect new, unseen flows. Specifically, when the dataset differs from the training dataset, it is referred to as cross-domain generalization performance. In a real-world scenario, the ML-based NIDS should possess the capability to detect various flows with different distributions. This is essential due to the inherent heterogeneity of network traffic.

4.7.1 *Generalization Evaluation Scenarios*

In this section, we explain various scenarios that highlight the generalization and cross-domain detection performance of the SSCL-NIDS from different perspectives and levels of difficulty.

Algorithm 2 : SSCL-NIDS Algorithm [58]

```
Input : Training data $\mathcal{X} \subseteq \mathcal{R}^M$, Batch size N, temperature $\tau$, anchor corruption
        rate $C_a$, positive pair corruption rate $C_p$, encoder network g, head
        network h
$t_a = \lfloor C_a \times M \rfloor$;
$t_p = \lfloor C_p \times M \rfloor$;
$B = \{1, ..., M\}$;
for sampled mini-batch $\{\mathbf{x}_i\}_{i=1}^N$ do
    forall $i \in \{1, ..., N\}$ do
        $A_i$: uniformly sample subset from B of size $t_a$;
        $P_i$: uniformly sample subset from B of size $t_p$;
        sample j uniformly from $\{1, ..., M\}$;
        if $j \notin A_i$ then
            $\mathbf{a}_i^j = \mathbf{x}_i^j$;
        else
            $\mathbf{a}_i^j = x_k^j$, where $x_k \sim \text{Uniform}(\mathcal{X})$;
        end
        let $\mathbf{z}_{2i-1} = h(g(\mathbf{a}_i))$;
        sample j uniformly from $\{1, ..., M\}$;
        if $j \notin P_i$ then
            $\mathbf{p}_i^j = \mathbf{x}_i^j$;
        else
            $\mathbf{p}_i^j = x_k^j$, where $x_k \sim \text{Uniform}(\mathcal{X})$;
        end
        let $\mathbf{z}_{2i} = h(g(\mathbf{p}_i))$;
    end
    forall $i \in \{1, ..., 2N\}$ and $j \in \{1, ..., 2N\}$ do
        $s_{i,j} = \frac{\mathbf{z}_i^T \mathbf{z}_j}{(\|\mathbf{z}_i\| \|\mathbf{z}_j\|)}$;
    end
    let $l(i, j) = -\log \frac{e^{s_{i,j}/\tau}}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} e^{s_{i,k}/\tau}}$;
    $L_{SSCL} = \frac{1}{2N} \sum_{k=1}^{N} \langle l(2k-1, 2k) + l(2k, 2k-1) \rangle$;
end
```

- **Scenario 1:** This scenario involves assessing the SSCL-NIDS's ability to generalize detection performance on previously unseen network flows, encompassing both benign and attack flows sourced from the same dataset as the training data.
- **Scenario 2:** Here, we assess the ability of the models to detect attack flows that were not present in the training dataset. The models are trained on benign flows from all five datasets and then evaluated on different previously unseen benign and attack flows sourced from these datasets.
- **Scenario 3:** To introduce an additional challenge to the evaluation process, this scenario entails training the model solely on benign flows from a single dataset. Subsequently, its ability to generalize detection performance across network traffic datasets is assessed by testing it on unseen flows from other datasets.

4.7.2 *Comparison Baselines*

In each of the generalization evaluation scenarios outlined in Section 4.7.1, the proposed SSCL-NIDS is benchmarked against supervised and unsupervised baseline models. The design of these baselines is detailed as follows.

- **Unsupervised Setup:** An AutoEncoder model is utilized as the unsupervised learning approach, focusing on optimizing data reconstruction. This model comprises an encoder and a decoder. The embedded data represents the compressed latent representations obtained from the encoder. Classification of flows using the AutoEncoder involves computing the distance of the new sample from the embedded training data (only benign flows). In this study, the AutoEncoder model is constructed with a three-layer encoder containing 64, 32, and 23 neurons, respectively, and a three-layer decoder containing 23, 32, and 45 neurons, respectively.
- **Supervised Setup:** A deep Multi-Layer Perceptron (MLP) model is employed as the supervised learning approach. This model consists of multiple layers designed to learn the connections between features and labels (benign/attack). Training this model requires labeled data for both benign and attack classes. The MLP model in this study comprises four layers with 64, 128, 64, and 32 neurons, respectively. A Sigmoid activation function is applied in the final layer to extract the score between 0 and 1 for each sample belonging to each class.

4.7.3 *Evaluation Datasets*

In this section, we provide information about the training and test datasets that are used to train and evaluate the SSCL-NIDS and supervised and unsupervised baselines, respectively. It is important to note that training the AutoEncoder and SSCL-NIDS models does not necessitate any attack flow samples. However, training the MLP model (our baseline for comparison) requires samples from both the benign and attack categories, as its learning process involves finding relationships between features of each class and the corresponding ground truth label.

- **Training Dataset for Unsupervised Models:** The training dataset used for SSCL-NIDS and AutoEncoder models includes 60% of only benign flows from each dataset. Depending on the specific generalization evaluation scenario outlined in Section 4.7.1, this can entail selecting 60% of the benign flows from each dataset individually (as in scenarios 1 and 3) or aggregating 60% of the benign flows from all datasets and training the models collectively (as in scenario 2).

- **Training Dataset for supervised Model:** For the MLP model, in scenarios 1 and 3, the training dataset includes 60% of the attack flows, alongside the training dataset used for the unsupervised models (SSCL-NIDS and AutoEncoder). In scenario 2, along with benign flows from all datasets, 60% of the attack flows from CICIDS17 are also included in the training dataset.

- **Test Dataset:** In scenario 1, evaluating the detection performance of the models on unseen flows sourced from the dataset used in training involves a test dataset comprising 40% unseen benign flows and all the attack flows from that dataset. For scenario 2, which explores cross-domain generalization, the model is assessed on 40% benign flows from different datasets, along with previously unseen attack flows. To enhance the evaluation difficulty in scenario 3, the model is tested on all the benign and attack flows from another network traffic dataset, which is different from the training dataset. It is important to note that for the MLP model in scenarios 1 and 3, evaluation is conducted on 40% of both attack and benign flows of the corresponding dataset. In scenario 2, the evaluation involves 40% of unseen benign flows and all attack flows except for CICIDS17, which accounts for only 40% of its attack flows.

4.7.4 *Evaluation Metrics*

The Area Under the Receiver Operating Characteristic curve (AUROC) is a crucial metric used to evaluate the performance of a binary classification model. It measures the model's ability to distinguish between positive (attack) and negative (benign) classes by comparing the true positive rate (TPR) to the false positive rate (FPR) across various threshold values, which are as follows. The TPR is represented in equation 11.

$$\mathsf{TPR} = \frac{\mathsf{TP}}{\mathsf{TP} + \mathsf{FN}} \tag{11}$$

where TP denotes true positive predictions and FN shows false negative predictions. Similarly, the FPR is determined by equation 12.

$$\mathsf{FPR} = \frac{\mathsf{FP}}{\mathsf{FP} + \mathsf{TN}} \tag{12}$$

where FP indicates false positive predictions and TN represents true negative predictions.

4.8 EVALUATION RESULTS

To perform the preprocessing and training, we utilize an Ubuntu server equipped with 250GB RAM and 4 GPUs (NVIDIA GeForce RTX 2080). The implementation is in Python, using the Scikit-learn, Pandas, and Pytorch libraries.

4.8.1 *SSCL-NIDS Training Hyper-parameters*

Following cross-validation on various hyper-parameters, a positive corruption rate of $C_p = 0.4$ is selected. The impact of this corruption rate on the final AUROC value will be shown in the evaluation results. The τ of the NTXent loss function in Equation 10 is set to 0.5. The embedding dimension remains the same as the original, i.e., $e_d = 45$. Moreover, the SSCL-NIDS trains for 500 epochs on the batch size of $bs = 2046$.

4.8.2 *Similarity Metric for Unsupervised Models*

This section is taken verbatim from [58].

To evaluate the embedded data learned by SSCL-NIDS and AutoEncoder, we compute a similarity score $sim(.)$ by calculating the cosine similarity between the embedding vector g_{test} of test data x_{test} and embedding vector g_m of all training data and finally take the maximum one. This calculation is performed using Equation 13.

$$sim(x_{test}) = \max_m\left(\frac{g_{test}^T g_m}{\|g_{test}\|\|g_m\|}\right) \tag{13}$$

According to Equation 13, it is possible to compute the directional similarity between the embedded benign training data and the embedded benign or attack data in the unseen test dataset. The expectation is that attack samples will exhibit a higher cosine similarity score, indicating that they are oriented in a different direction compared to benign samples.

4.8.3 *Impact of the Corruption Rate on Detection Performance*

This section is taken verbatim from [58].

As explained in Section 4.6.2, the anchor corruption rate (C_a) and positive pair corruption rate (C_p) are fundamental in SSCL-NIDS architecture. While C_p controls the similarity of constructed positive pairs, C_a introduces additional noise to the original data, challenging the SSCL-NIDS model, leading to an improvement in its detection performance generalization. Through our experiments, we observed that selecting C_a greater than 0.2 reduces the model's detection performance due to excessive noise added to the original data. Therefore, we set $C_a = 0.2$ for all evaluation results. Additionally, varying the value of C_p influences the corruption of a larger subset of features, generating dissimilar positive pairs and impacting the optimization process. A higher C_p makes the optimization task more challenging, while a

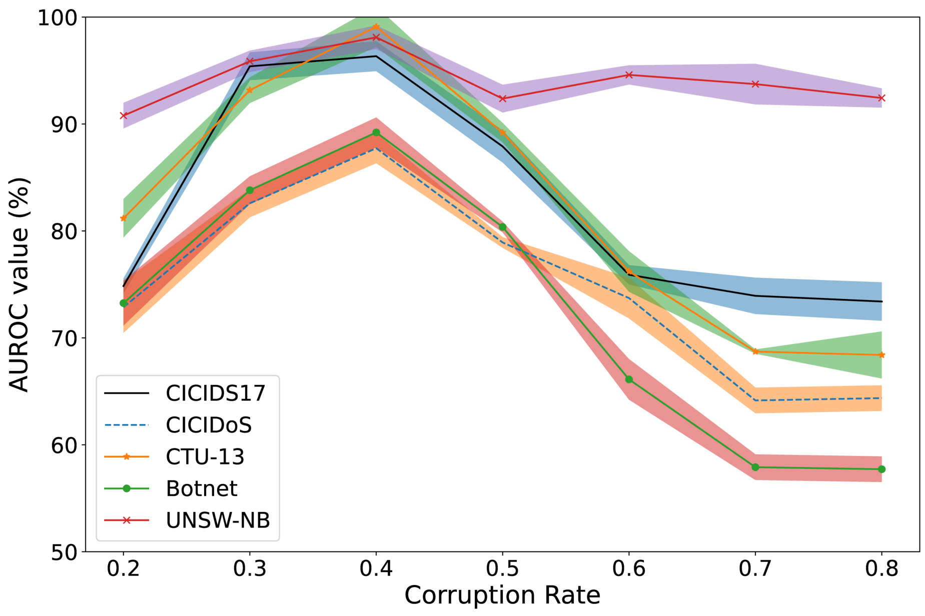

Figure 11: Impact of positive pair corruption rate (C_p) on detection performance. The experiment is carried out over three independent runs, and the figure depicts the mean value and standard deviation for each dataset. *The figure is extracted from [58].*

smaller C_p results in a less robust representation. Figure 11 illustrates the impact of different C_p values on the final detection performance, measured through AUROC values, with the experiment conducted three times independently. According to the findings presented in Figure 11, a C_p value of 0.4 consistently yields the highest detection performance across all datasets. In contrast, a very small C_p value, specifically $C_p = 0.2$, results in a lower AUROC value than $C_p = 0.3$ and $C_p = 0.4$. Therefore, we choose $C_p = 0.4$ as the optimal corruption rate for the final evaluation results of SSCL-NIDS.

4.8.4 *Scenario 1: Detection Performance on Intra-Dataset Flows*

In this scenario, we assess the performance of our proposed SSCL-NIDS model using a test dataset derived from the same source as the training dataset, referred to as intra-dataset flow samples. In this experiment, the unsupervised models, including SSCL-NIDS and AutoEncoder, are trained solely on 60% of benign flows from each dataset. The remaining 40% of the test data comprises the remaining benign flows and all attacks from the corresponding dataset. Conversely, for the supervised learning model, MLP, training involves 60% of both benign and attack flows from each dataset, with an evaluation conducted on the remaining 40% of benign and attack data. Table 8 illustrates the results of this experiment across all five distinct datasets. The results indicate that, as expected, supervised learning exhibits better detection performance compared to other unsupervised models due to its training on both benign and attack flows of each dataset individually. However, in contrast to the Au-

Table 8: Evaluating the detection performance of *SSCL-NIDS* and comparing it with baseline models, including an unsupervised model (*AutoEncoder*) and a supervised model (*MLP*). Their detection performance is evaluated on previously unseen data, maintaining the same distribution as the training data, referred to as intra-dataset flows.

Dataset	AUROC Value (%)		
	SSCL-IDS	Unsupervised Model	Supervised Model
CICIDS17	92.78	83.51	94.64
CICDoS	83.25	76.11	86.54
CTU-13	93.92	84.19	99.13
Botnet	88.82	81.92	90.96
UNSW-NB	92.07	82.80	88.98

toEncoder, the proposed SSCL-NIDS demonstrates better performance, with results that are close to those of the supervised learning model. This comes from the differences between the learning process of self-supervised learning and AutoEncoder. The results show that SSCL-NIDS could learn more abstract representations of the benign flows.

4.8.5 *Scenario 2: Detection Performance on Cross-Domain Attacks*

To assess the cross-domain detection capabilities of the ML models, we train the unsupervised models using 60% of benign flows from all datasets and evaluate them on the remaining unseen flows, comprising 40% benign flows and all attack instances. Additionally, in the supervised learning model, we also include training on 60% of attack flows from the CICIDS17 dataset. Table 9 demonstrates the detection performance of these models across all five datasets.

Table 9: Evaluating the detection performance of *SSCL-NIDS* and comparing it with baseline models, including an unsupervised model (*AutoEncoder*) and a supervised model (*MLP*). This evaluation is conducted on unseen attack flows, which are not available in the training dataset. In this scenario, ML models are trained on all the benign flows of all the datasets. *The table is extracted from [58].*

Dataset	AUROC Value (%)		
	SSCL-IDS	Unsupervised Model	Supervised Model
CICIDS17	96.54	70.79	**97.25**
CICDoS	**87.73**	72.71	57.14
CTU-13	**99.85**	77.61	72.77
Botnet	**89.21**	73.48	60.64
UNSW-NB	**98.32**	65.64	60.59

As indicated in Table 9, SSCL-NIDS demonstrates the highest cross-domain detection performance compared to both supervised and unsupervised learning baseline models, with the exception of the CICIDS17 dataset. For this dataset, where the supervised learning model was trained on its attack flows, SSCL-NIDS achieves the second-best result, closely trailing the supervised model (MLP). Therefore, for CICIDS17, the evaluation is like the first scenario. However, it's worth noting that training on labeled data may not entirely show real-world conditions. The supervised model's detection performance on other datasets reflects its cross-domain generalization performance, which is the lowest, as expected. In contrast, the proposed SSCL-NIDS model exhibits up to 27% better detection performance compared to the supervised model across all datasets. Moreover, it outperforms the unsupervised model by up to 15% across all datasets.

We argue that the enhanced generalization of SSCL-NIDS is attributable to differences in its learning processes. Specifically, contrastive learning within SSCL-NIDS encourages the model to aggregate the representations of samples with similar semantics in the embedding space. This approach, as supported by prior research [31, 139], facilitates the acquisition of more effective representations related to the semantics of the training data. Consequently, it enables the model to learn a more efficient representation of benign flows.

4.8.6 *Scenario 3: Detection Performance on Cross-Domain Flows*

In Section 4.8.4, we evaluate the detection performance of the models on samples originating from the same source as their training dataset. The results indicate that SSCL-NIDS closely follows the performance of the supervised learning model, although it trained only on benign flows. Moving to Section 4.8.5, we assess the models' detection performance on cross-domain attack samples, where they are trained on 60% of all benign flows. In that scenario, SSCL-NIDS outperforms other models, while the supervised learning model exhibits the lowest performance, as expected.

This leads to two questions:

- How do the results change when models are trained solely on benign flows from one dataset instead of all datasets?
- What impact does adding new flows from different datasets to the training dataset have on the final detection results?

In this section, we address the first question by training the models on 60% of benign flows from only one dataset at a time, while evaluating them on all other datasets. This scenario poses a more challenging task for evaluating generalization since the models have not been exposed to benign flows from other datasets, as well. Consequently, both benign and attack flows are unseen, new, and different for the ML models.

The results depicted in Figure 12 illustrate the outcomes of the scenario. The upper figure represents the results of SSCL-NIDS, while the lower figure depicts the

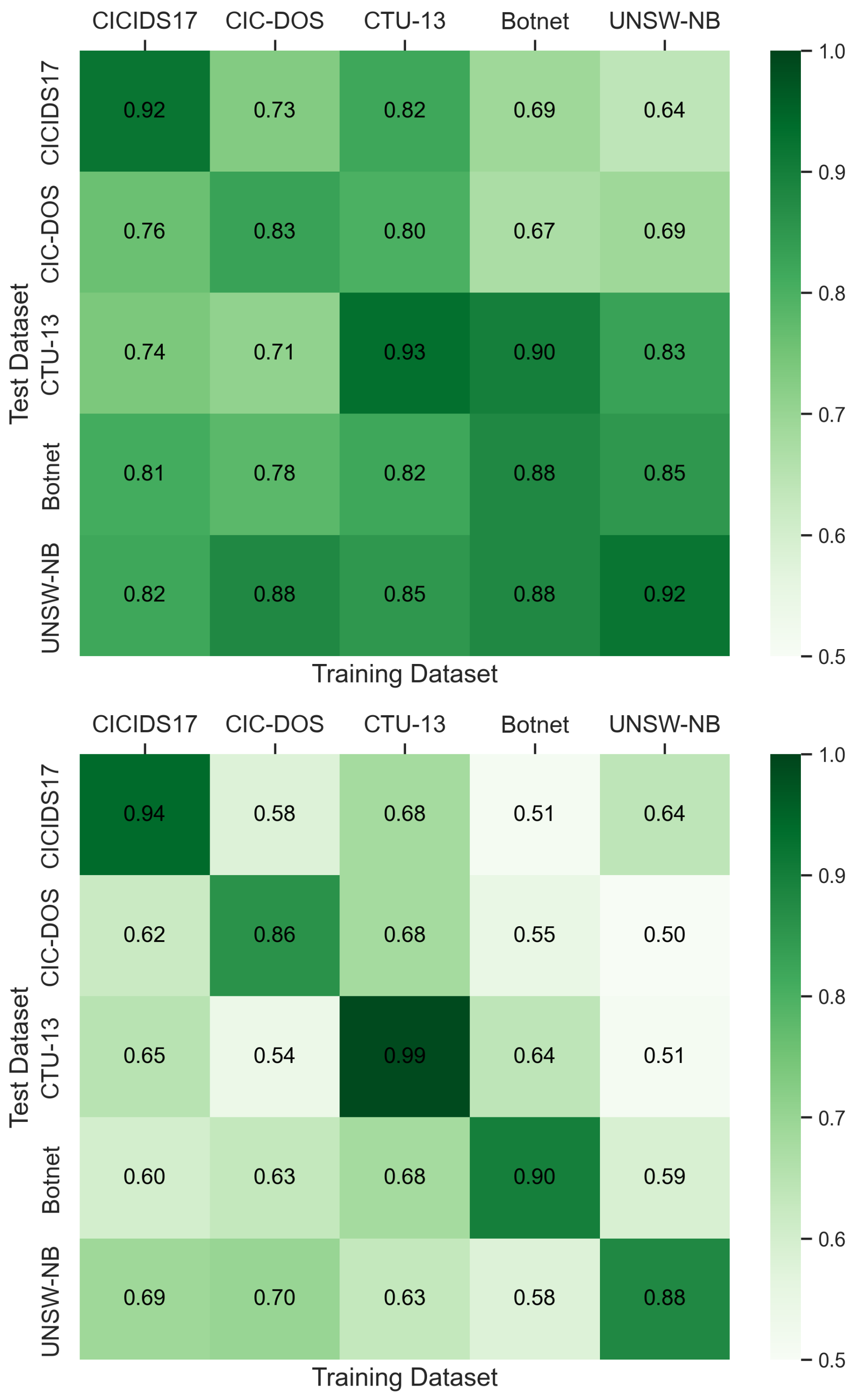

Figure 12: Comparison of detection performance generalization between *SSCL-NIDS* (upper) and the *supervised* ML-IDS (lower). Both models are trained on one dataset and evaluated on other network traffic datasets. Diagonal values indicate evaluation on a network traffic dataset similar to the training set. *The figure is extracted from [58].*

results of the supervised model. The diagonal values are representing scenario 1 results, where each model is trained on a dataset and evaluated on unseen data of the same dataset. A comparison of these figures highlights the improved generalization of SSCL-NIDS compared to the supervised learning model. Notably, the detection performance of the MLP model on CICIDS17 and CICDoS, when trained on Botnet, is approximately 50%, akin to random prediction.

Moreover, training SSCL-NIDS with certain datasets, such as CTU-13 and CICIDS17, enables the model to acquire more abstract representations of benign flows, resulting in better cross-domain detection performance on other datasets. For instance, evaluation with test datasets like UNSW-NB demonstrates detection performance exceeding 80% with the trained model from any of the datasets. This suggests that attack types exhibit different behaviors from benign flows, rendering them more easily detectable with high accuracy. However, even though SSCL-NIDS achieves high detection performance on the UNSW-NB dataset when trained on any dataset, MLP fails to achieve high detection performance on it when trained with datasets other than UNSW-NB.

Furthermore, as shown in Figure 12, when the SSCL-NIDS model is trained on dataset X and evaluated on dataset Y, its detection performance differs from when it is trained on dataset Y and evaluated on dataset X. Hence, each dataset reveals distinct aspects of benign flows and the SSCL-NIDS can learn different information about benign flows by training on them. This observation leads us to proceed with the next evaluation, aimed at assessing the impact of incorporating new datasets into the training data on the final detection performance.

4.8.7 *Impact of Adding New Dataset to the Training Data*

This section is taken almost verbatim from [58].

In this section, we address the second question outlined in Section 4.8.6, which explores the impact of adding new flows from different datasets on the final detection and generalization performance of the SSCL-NIDS. The result of this evaluation is illustrated in Figure 13. The number of datasets increases following the sequence outlined in Table 8. To clarify, the initial dataset (One DS) is *CICIDS17*, with *CICDOS* added in the second dataset (Two DS), and so on.

Notably, *CICDOS* shows an increase in AUROC value when included in the Two DS training, and a similar improvement is observed for *CTU13* in the Three DS training with the addition of its dataset. Likewise, the detection performance of *UNSW-NB* shows a high value when the *SSCL-IDS* is trained with all datasets, including its training dataset.

Our findings show that incorporating additional benign flows from various datasets encourages the model to learn a more comprehensive understanding of benign traffic, thereby enhancing its generalization and detection performance. Furthermore, training the model on diverse benign flows sourced from multiple datasets facilitates the transfer of this pre-trained model to other network classification tasks. The same concept is studied in the area of Large Language Models (LLM) [125].

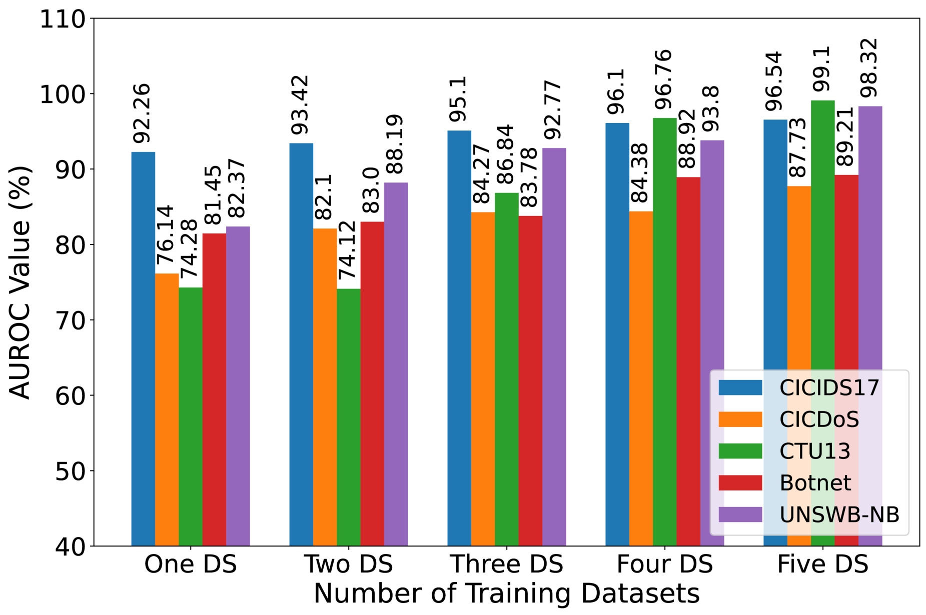

Figure 13: Comparing AUROC values for each dataset when the number of training datasets increases. In the x-axis of the figure, 'DS' refers to the dataset. As expected, when the dataset is added to the training data, its AUROC value increases. The number of datasets increases following the sequence outlined in Table 8. *The figure is extracted from [58].*

4.8.8 *t-Distributed Stochastic Neighbor Embedding (t-SNE)*

In this experiment, which is also available in [58], we employ t-Distributed Stochastic Neighbor Embedding (t-SNE), a dimensionality reduction technique, to visualize the embedding data produced by the SSCL-NIDS in a lower-dimensional space [113]. This approach enables us to evaluate the quality of the learned embeddings, demonstrating how well the model has captured the inherent structure of the data. At its core, t-SNE aims to map high-dimensional data points to a lower-dimensional space while preserving local similarities between points as accurately as possible. This is achieved through a two-step process: first, computing pairwise similarities between data points using a Gaussian kernel to measure similarity in the high-dimensional space. The similarity p_{ij} between data point x_i and x_j is computed according to equation 14 [113].

$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)} \tag{14}$$

Where σ_i, is the variance of the Gaussian kernel which determines the effective number of neighbors for each data point.

Next, t-SNE defines conditional probabilities q_{ij} in the lower-dimensional space. It aims to represent pairwise similarities p_{ij} in the high-dimensional space with conditional probabilities q_{ij} in the lower-dimensional space. These conditional probabilities are determined by a t-distribution as shown in equation 15, aiming to closely

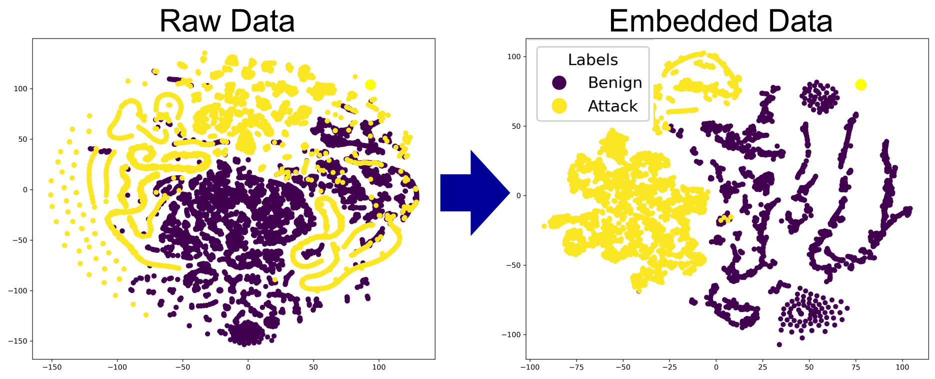

Figure 14: Comparing the embedded data with the raw data of the CICIDS17 dataset in lower-dimensional space. The t-SNE plot of the embedded data (the right one) depicts a separation between data points belonging to different classes. *The figure is extracted from [58].*

match the pairwise similarities in the lower-dimensional space to those in the high-dimensional space [113].

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}} \tag{15}$$

where, y_i and y_j are the low-dimensional representations of x_i and x_j, respectively.

The optimization objective of t-SNE involves minimizing the Kullback-Leibler divergence between the distributions of pairwise similarities in the high-dimensional and lower-dimensional spaces. This is achieved by minimizing the cost function shown in equation 16 using gradient descent or other optimization techniques [113].

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{16}$$

where P_i and Q_i are the distributions of pairwise similarities for data point x_i in the high-dimensional and lower-dimensional spaces, respectively.

Figure 14 shows the t-SNE plots for both the raw data and the embedded data (output of the SSCL-NIDS) within the CICIDS17 dataset. Clustering of similar examples in the t-SNE plot indicates that the proposed SSCL-NIDS has effectively captured relevant features and relationships within the data.

Notably, in the plot with embedded data, the data points of each class are closely grouped, exhibiting clearer separation and fewer overlaps compared to the t-SNE plot of the raw data. This improved separation enhances the model's ability to distinguish between attack and benign flows, thereby improving overall detection performance.

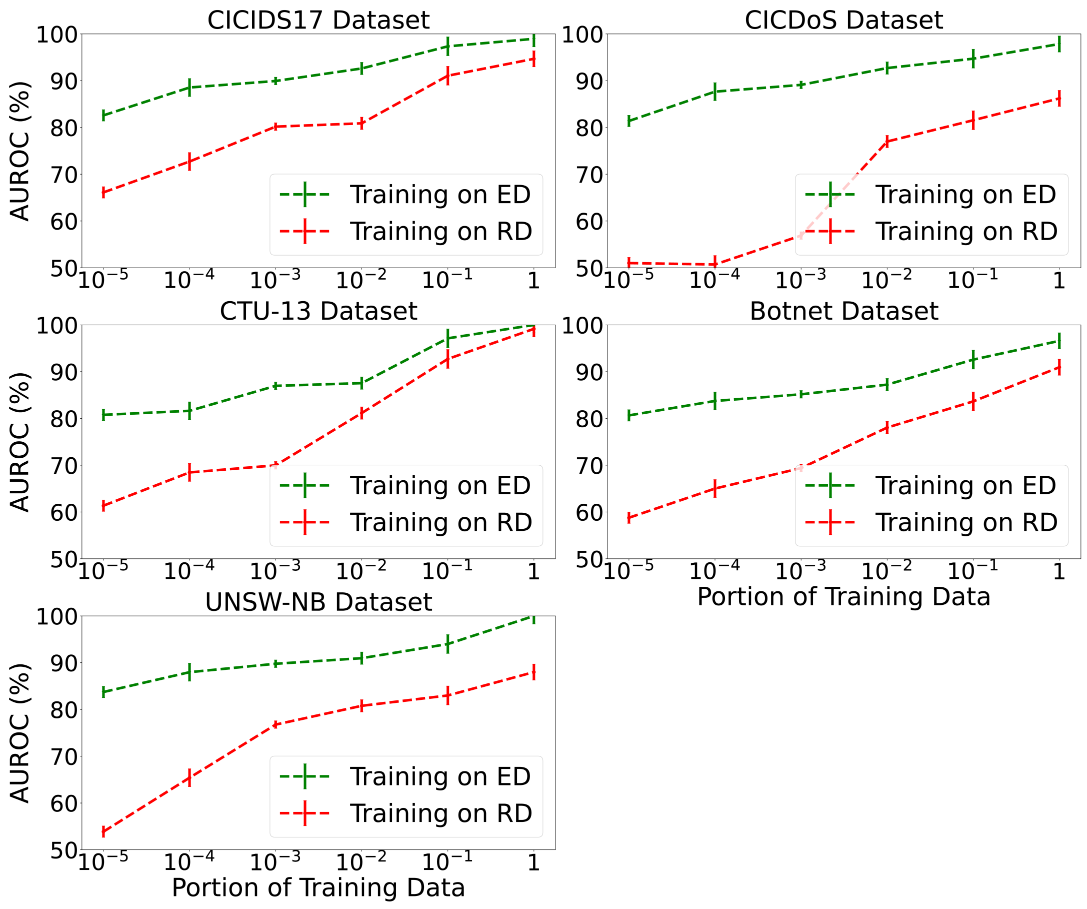

Figure 15: Comparing label efficiency for the benign/attack classification task, 'ED' represents embedded data, reflecting the detection performance when transfer learning is employed. On the other hand, 'RD' denotes raw data, indicating the results of supervised ML-IDS training. *The similar figure is in [58].*

4.8.9 *Sample Efficiency of SSCL-NIDS for Transfer Learning*

Transfer learning, as defined in Section 4.4, is a machine learning technique that leverages the knowledge acquired by a pre-trained model to tackle new tasks, especially in situations where labeled data is scarce or difficult to obtain.

In this section, we explore the transferability of the SSCL-NIDS, specifically examining the number of labeled samples required to achieve high detection performance using transfer learning. To accomplish this, we utilize the SSCL-NIDS as a pre-trained model and fine-tune it with an MLP model possessing the same configuration outlined in Section 4.7.2, utilizing various proportions of flow samples. The results of this investigation are presented in Figure 15. For comparative analysis, we also train the same MLP model using raw data (RD) and assess its performance in comparison to when it is trained on SSCL-NIDS representations, referred to as embedded data (ED).

To evaluate sample efficiency for each dataset, labeled samples are incrementally added to the training dataset, beginning from a portion equivalent to 10^{-5} of the original dataset size. This initial portion comprises only 7 to 20 labeled samples, depending on the size of the dataset. As illustrated in Figure 15, across all datasets, the AUROC values indicate higher detection performance when the SSCL-NIDS is fine-tuned for the classification task. This performance gap is bigger when only a small number of labeled samples are used for training. For instance, the detection performance for the CICDoS and UNSW-NB datasets, when trained on 20 or fewer labeled samples of raw data, is similar to random guessing (AUROC = 50%). In contrast, using pre-trained embeddings yields AUROC values exceeding 80%.

Furthermore, it's noteworthy that higher detection performance is achieved when utilizing all labeled samples for training with the pre-trained SSCL-NIDS compared to training with raw data.

4.8.10 *Comparing SSCL-NIDS with State-of-the-Art Approaches*

In this section, we conduct a comparative analysis of our proposed SSCL-NIDS with other related works that employ SSL as a basis for ML-based NIDS. This comparison is based on the reported values provided by these studies. It is important to note that variations in preprocessing pipelines among different works may influence their final results.

Table 10 comprises two categories: supervised SSL methods (first category), where label information was utilized in pretext tasks or augmentation approaches, and unsupervised SSL methods (second category), where the flows' ground-truth labels were not used. The evaluation metrics include the AUROC score and F1-score, en-

Table 10: Comparison of SSCL-NIDS with SSL-based State-of-the-Art approaches. Notably, supervised SSL relies on labeled data (Benign/Attack) for training, whereas unsupervised models operate without such labels. *The same table is in [58].*

Related Work	**AUROC / F1-Score (%)**				
	CICIDS17	CICDoS	CTU-13	Botnet	UNSW-NB
Supervised SSL:					
RLB-CL [110]	- / 90.72	-	-	- / 85.65	- / 89.42
ConFlow [105]	- / 99.96	-	-	- / 99.16	-
CLDNN [177]	- / 99.96	-	-	- / 99.47	- / 92.91
TS-IDS [126]	98.80 / 99.55	-	-	97.14 / 96.67	99.86 / 99.75
Unsupervised SSL:					
BYOL [167]	96.0 / 95.48	-	-	97.0 / 98.46	88.0 / 92.41
Anomal-E [26]	- / 90.72	-	-	-	- / 92.18
SSCL-NIDS (ours)	96.54 / 97.73	87.73 / 95.47	99.85 / 98.32	89.21 / 97.16	98.32 / 99.43

abling comprehensive comparisons between SSCL-NIDS and other related works. While some related studies assessed their models using the NSL-KDD network traffic dataset, we could not do the same because the network traffic in pcap format was not available. To enhance the flexibility of SSCL-NIDS for evaluating generalization on new datasets, we implemented a preprocessing pipeline (outlined in Section 4.6.1) capable of handling traffic files. Additionally, no SSL-based approach has been evaluated on the CICDoS and CTU-13 network traffic datasets based on available information.

As indicated in Table 10, some supervised SSL techniques that leverage label information to generate positive and negative pairs during data augmentation exhibit better detection performance compared to SSCL-NIDS, which does not use label information for forming positive pairs [58]. The details for each related work are available in Section 2.6.1. The difference in their detection performance on the CICIDS17 network traffic dataset is at most around 2%, on the UNSW-NB dataset around 1%, and on the Botnet dataset it is approximately 8%. Note that SSCL-NIDS outperforms the majority of supervised SSL methods on the UNSW-NB network traffic dataset.

Moreover, SSCL-NIDS outperforms all other unsupervised SSL-based related works, except for *BYOL*, specifically on the Botnet dataset. A comparison of SSCL-NIDS with *BYOL* on other datasets shows that our model outperforms it across other network traffic datasets. This difference in detection performance could stem from various factors, including disparities in preprocessing pipelines, such as variations in the utilized flow features or differences in the augmentation process for creating positive pairs. In fact, these factors can result in the model providing better insights into the benign flows in one dataset compared to other datasets.

To assess the generalization and cross-domain detection performance, we conduct evaluations of our proposed SSCL-NIDS on a broader range of network traffic datasets compared to existing approaches. For example, the CTU-13 network traffic dataset encompasses botnets involved in multi-stage attacks (MSA), which are known for their challenging detection characteristics [99]. Additionally, the CICDoS network traffic dataset includes SlowDoS attacks, which can mimic benign flow patterns. Our approach, SSCL-NIDS, demonstrates high detection performance on these datasets, underscoring its effectiveness in detecting sophisticated attack flows.

4.9 SUMMARY

In this chapter, we delve into the challenges associated with supervised ML-based NIDSs. These challenges include their reliance on annotated network traffic datasets, the risk of overfitting due to imbalanced network traffic datasets, and the scarcity of labeled network intrusions. These issues collectively constrain supervised learning models to train on limited network traffic datasets with a restricted number of network intrusions. Consequently, this limitation can result in degraded detection performance when the model encounters previously unseen network flows with potentially different distributions from the training dataset. Similar challenges apply to unsupervised learning, as it optimizes for features that may only be effective for the training dataset.

However, in real-world scenarios, network traffic is heterogeneous, and new network intrusions continually emerge. Therefore, there is a necessity for designing a generalized ML-based NIDS capable of achieving high detection performance across various unseen network traffic datasets, ideally demonstrating high cross-domain detection performance. To tackle this challenge, we introduce SSCL-NIDS [58] in this chapter, which utilizes self-supervised contrastive learning to learn abstract representations of flows and exclusively trains on benign flows, reducing the dependency on annotated datasets.

Our results demonstrate that, compared to supervised and unsupervised baseline models, our proposed SSCL-NIDS achieves higher cross-domain detection performance. Furthermore, we investigate the transferability of SSCL-NIDS and find that even with a small number of annotated samples (less than 20), SSCL-NIDS achieves detection performance exceeding 80%. This suggests the potential for utilizing pretrained SSCL-NIDS for online learning to rapidly adapt to new distributed network flows while maintaining high detection performance.

5 INTEGRATING MACHINE LEARNING IN PROGRAMMABLE NETWORKS

In this chapter, Section 5.8.9 is taken almost verbatim from [61].

Software-defined networking (SDN) revolutionizes network control by separating the data plane and control plane functionalities, offering enhanced automation and flexibility compared to traditional networks [153]. This architecture facilitates efficient resource utilization, greater programmability, and heightened flexibility, addressing the limitations of legacy networking [78]. More details about SDN can be found in Section 2.5.2.

The centralized design of the SDN control plane exposes it to diverse cyber-attacks, such as Denial of Service (DoS) or Distributed DoS (DDoS) attacks, aiming to disrupt network operations [29]. These attacks can directly target the control plane or indirectly impact it by flooding the data plane [83]. When attackers target the control plane through the data plane, excessive flows may flood the switch, depleting its flow table memory and preventing the acceptance of new flows. Additionally, rapid influxes of packet flows can exhaust the switch's buffer, causing it to forward all buffered packets to the control plane, potentially overwhelming its bandwidth [45].

Therefore, equipping SDN with an effective NIDS is essential. ML-based NIDSs emerge as a suitable choice due to their ability to distinguish between attack and benign flows through training on flow features. Unlike signature-based NIDSs, they excel in classifying new flows, not just existing ones. In these approaches, a flow comprises all packets that share identical characteristics, such as source and destination IP addresses, L4 ports, and protocols [157].

Given the intensive computational requirements for training ML models, integrating an ML-based NIDS into the SDN control plane, which provides substantial computational resources, is entirely viable [115]. However, for each flow to be classified using this system, flow features must traverse from the data plane to the control plane, where the ML-based NIDS resides [61]. Following this, the control plane can establish appropriate network policies based on the classification outcomes (attack-/benign) for each flow, subsequently implementing them in the data plane. Nevertheless, this process of transmitting flows and populating policies can increase network load and overwhelm the control plane.

Additionally, the rapid detection of attack traffic flows is often mandated to occur within a short timeframe according to security standards [180]. However, the flow detection process performed by the ML-based NIDS deployed in the control plane does not operate at a line rate.

SDN has evolved with programmable switches, allowing the programming of data plane behavior using languages like Programming Protocol-Independent Packet Processors (P4)[18, 19]. More details about programmable switch structure can be found in Section 2.5.1. Implementing an IDS in the data plane using these switches can reduce latency and improve intrusion detection speed. However, training an ML-based IDS in the data plane faces challenges due to switch resource constraints. Integrating a pre-trained ML model into the P4 switch's MA pipeline enables traffic classification but may result in decreased detection performance and increased misclassification risk, impacting network QoS and security.

To maintain high detection performance while minimizing latency and resource usage, alternative approaches are necessary. Our proposed Collaborative ML-based NIDS (CML-IDS) leverages ML-based NIDS deployment in both the data plane and control plane to achieve these goals.

5.1 DEPLOYING MACHINE LEARNING MODEL IN THE CONTROL PLANE

Within SDN, the control plane has a global network view, which facilitates traffic collection and monitoring. Moreover, it has sufficient computational resources for implementing a preprocessing pipeline and training ML models; therefore, the ML-driven approaches are generally implemented in the control plane [86].

Definition — Here, ***CP-IDS*** is an ML-based NIDS deployed in the control plane that integrates dataset preprocessing, ML model training, and decision-making modules. The sufficient computational resources available in the control plane allow for the implementation of more complex ML models [172].

Deploying various ML models, including supervised, unsupervised, and ensemble techniques, in the control plane presents an opportunity. This diversity, along with utilizing ML models with more learnable parameters, has the potential to enhance detection performance.

Nevertheless, forwarding a large volume of flows to the control plane for classification could result in increasing network load and overwhelm the control plane. Moreover, the detection speed may fall short of the line rate.

5.2 DEPLOYING MACHINE LEARNING MODEL IN THE PROGRAMMABLE DATA PLANE

To address the limitations of CP-IDS, the concept of deploying ML-based NIDS within the programmable data plan has been introduced [23, 98, 171].

Definition — Here, ***DP-IDS*** is an ML-based NIDS deployed in the programmable data plane. The structure of the programmable switch exhibits similarities with tree-based ML models, enabling their deployment in these switches [171].

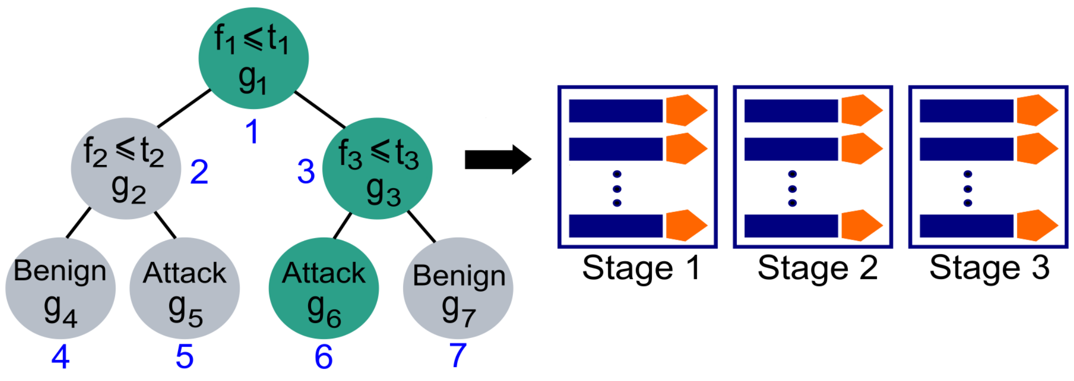

Figure 16: An example to illustrate the structural similarity between Decision Tree classifiers and Match-Action pipelines in P4 switches. *The figure is extracted from [61].*

The programmable data plane, facilitated by the P4 language [18], allows for adaptable packet processing.

In programmable switches, packet processing begins with a parser, which directly extracts headers from incoming packets at the ingress port [104]. These packets are then categorized into flows using a flow ID generated by hashing 5-tuple fields, including source and destination IP addresses, port numbers, and the transport layer protocol. Each flow entry, along with its corresponding flow ID index, is stored in a flow buffer. Upon packet arrival, relevant features undergo continuous updates in registers until either the last packet in the flow arrives or the arrival time exceeds the timeout threshold. While these registers are accessible from both the data plane and control plane, the switch's limited resource capacity imposes constraints, allowing only a finite number of registers in a programmable switch. Furthermore, the P4 language's limitations, such as the absence of support for iterative constructs and certain mathematical operations like division, logarithm, and exponentiation, restrict the ML models implementable within the programmable switch using the P4 language. These constraints confine the deployed DP-IDS to lightweight designs aligned with the switch's pipeline. However, integrating a lightweight ML model may compromise overall detection performance.

5.2.1 *Similarity between Random Forest and Match-Action Pipeline*

The RF classifier, detailed in Section 2.4.1, is an ensemble ML model composed of multiple DT models. In the RF model, decision-making relies on majority voting over the decisions made by individual DTs.

DT's classification structure is similar to the MA table structure in a P4 programmable switch, making it well-suited for integration into the switch as the DP-IDS. To illustrate the similarity between the RF model and MA tables, we present an example of a two-level DT model in Figure 16. In each node, the flow feature f_i, threshold t_i for data splitting, and Gini impurity value g_i represents purity measurement. We highlight the classification path (1-3-6) to explain the decision-making process of a DT model [61]. Depending on the feature value comparison with the threshold at each node, the path proceeds left if the condition is met. This decision-making process continues, classifying the input flow as attack or benign. DT classification relies solely on feature value comparisons without complex mathematical operations.

In a programmable switch, packets undergo processing through the MA pipeline for flow classification. This pipeline comprises sequential stages where packets are processed based on predefined MA tables containing keys for matching and actions to execute. Entries in these tables, populated from the control plane, define keys and actions executed upon match. While each table can have multiple entries, it is executed only once through the pipeline. Additionally, output from the previous stage can serve as input in the current stage, facilitating sequential processing.

Feature comparison execution in a DT classifier occurs once at each level along a classification path, similar to MA table operation at each pipeline stage. Each DT level, except the root node, contains multiple intermediate nodes or leaves. Entries in Match-Action (MA) tables can construct nodes at each DT level. The outcome of feature comparison determines the next level's node choice, aligning with sequential MA pipeline execution.

5.2.2 *Integrating a Random Forest into a Match-Action Pipeline*

In this section, we discuss how the Decision Tree (DT) depicted in Figure 16 can be integrated into a MA pipeline, as shown in Figure 17. Within the MA pipeline, MA tables interpret both tree nodes and leaves, utilizing their sequential execution to construct a classification path within the programmable switch.

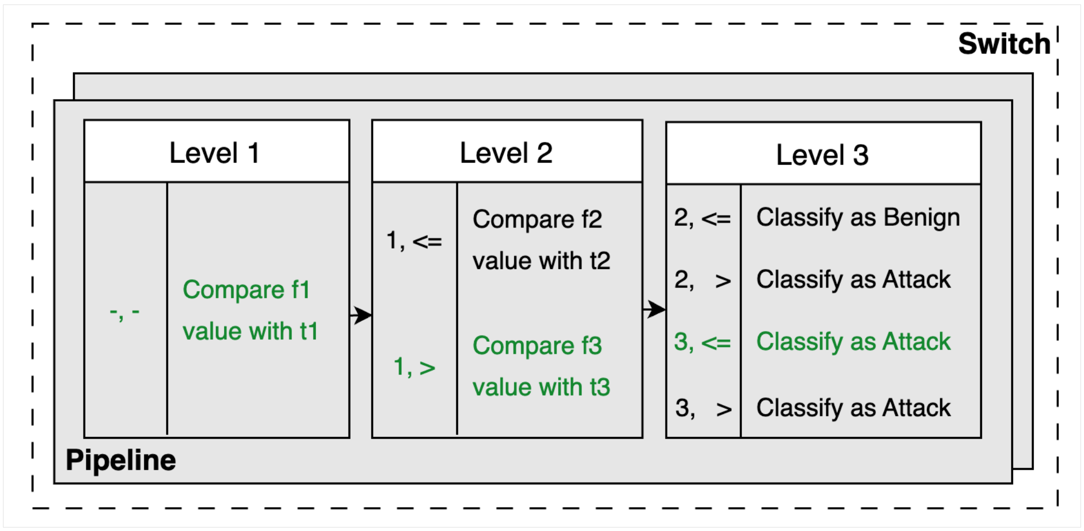

Figure 17: Match-action table entries which are established to construct the DT depicted in Figure 16. In this representation, "-" indicates that there is no value for a key to compare, as the root node does not have a previous node. It is important to note that the keys of the root node are always a match. The left column in each MA table represents keys, while the right column represents the required actions for the corresponding key.

During traversal along the classification path within a DT, two operations occur. At the root and intermediate nodes, the action involves comparing the feature value with the threshold and determining the next node. At leaves, the classification concludes, and the incoming flow is classified as benign or attack, requiring an action to classify accordingly. To determine the executed action, the previous node ID and the result of the previous feature comparison can serve as keys in the MA table. The previous node ID indicates the reached classification step, while the previous feature comparison result directs the classification path.

As depicted in Figure 16, the sample tree consists of three levels, necessitating three stages in the MA pipeline, as illustrated in Figure 17. Each stage executes an MA table with keys and actions. Stage 1, with the level 1 MA table, contains one table entry. Stage 2, with the level 2 MA table, includes two table entries, while stage 3 has four table entries due to the four leaves in the tree structure. The sample classification path involves nodes 1, 3, and 6, with feature values f_1 and f_3 used for flow classification, requiring three table entries to construct the path.

A Random Forest (RF) classifier comprises multiple DTs. The number of tables needed to interpret an RF equals the total number of levels in all trees, while the number of table entries equals the sum of all nodes in the trees. However, the hardware programmable switch supports a limited number of stages. Therefore, the complexity of the RF switch should align with available hardware resources.

5.3 CONTRIBUTIONS

Section 5.1 and Section 5.2 explored the deployment possibilities of CP-IDS and DP-IDS, each with its own set of advantages and drawbacks. For instance, CP-IDS offers to use a more complex ML model with more trainable parameters, which leads to a more accurate classification, or DP-IDS can classify each flow in a line rate and avoid forwarding flows to the control plane. However, neither of them can fully meet the requirements for achieving high detection performance, speed, and low network load simultaneously.

This chapter introduces a novel collaborative framework aimed at enhancing detection performance compared to DP-IDS alone, while also improving detection speed and reducing network load compared to CP-IDS alone.

5.4 COLLABORATIVE MACHINE LEARNING IN SOFTWARE-DEFINED NETWORKING

The Collaborative ML-based Network Intrusion Detection System (CML-IDS) comprises two distinct machine learning (ML) models: DP-IDS and CP-IDS. DP-IDS is a Random Forest (RF) model deployed in the data plane, while CP-IDS is an ensemble ML model consisting of RF, XGBoost, and MLP models deployed in the control plane. To effectively utilize both models, as discussed in Section 5.3, the initial step involves determining when to employ DP-IDS and when to utilize CP-IDS. This determination is based on defining a threshold for the model confidence (MC) of DP-IDS. If the

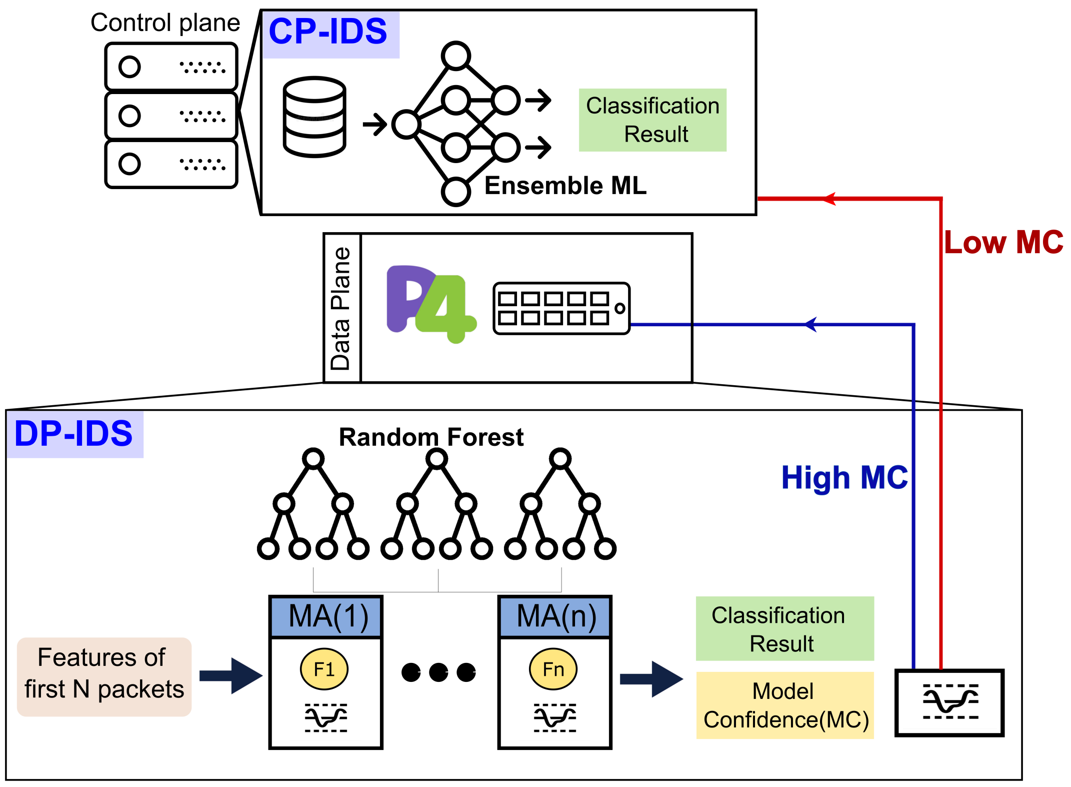

Figure 18: The general architecture of CML-IDS (Collaborative ML-based IDS) involves the deployment of DP-IDS in the programmable data plane, which works in collaboration with CP-IDS deployed in the control plane. If the model confidence of DP-IDS fails to meet the predefined threshold of MC_{thr}, then CP-IDS classifies the flow. Otherwise, DP-IDS classifies the flow. *The figure is extracted from [61].*

MC for a classified flow does not exceed the threshold (MC_{th}), the flow is forwarded to the control plane for classification using CP-IDS. The overall architecture of the CML-IDS framework is depicted in Figure 18. For successful collaboration between these components, additional modules within the control plane and data plane are required.

5.4.1 *Required Modules in the Control Plane*

This section provides an explanation of the Machine Learning, P4 Program Generator, and Data Plane Control modules, which are integrated into the control plane.

Machine Learning Module

The Machine Learning module carries several responsibilities. Firstly, it preprocesses the network traffic dataset, encompassing tasks such as converting PCAP traffic files to network traffic features suitable for training ML models and performing feature selection. Within this module, the ML models used for both the DP-IDS and CP-IDS are trained. Furthermore, it manages the final prediction of flows that the DP-IDS could not classify.

P4 Program Generator Module

The P4 Program Generator is one of these crucial modules for dynamically generating P4 code snippets and MA table entries using the trained embedded RF classifier. Whenever a new RF classifier is trained for embedding as the DP-IDS, the P4 Program Generator automatically generates the necessary P4 code. This code is then merged with predetermined components to create a comprehensive P4 program, which is embedded into the programmable switch.

Data Plane Control Module

Another essential module is the Data Plane Control Module, specifically designed to interface with the data plane. This module is responsible for installing the generated P4 program into the data plane and populating the entries of an MA table. Additionally, it handles forwarding the final decisions of the CP-IDS to the programmable data plane for further actions, such as either forwarding the flow or blocking it. The PacketIO sub-module facilitates packet transfer between the switch and controller, including directing a flow to the controller and delivering the predicted label to the switch.

5.4.2 *Programmable Switch in the Data Plane*

The programmable switch within the programmable data plane operates using the P4 program, which is installed via the Data Plane Control module in the control plane and PacktIO sub-module. It manages incoming packets by parsing their packet headers. Additionally, it is responsible for defining and identifying flows based on the hash value obtained using 5-tuple fields. Each identified flow is stored in the flow buffer, and its features are updated based on the arrival packet information. These features are crucial for the classification of each flow using the DP-IDS. If the confidence of the DP-IDS falls below the predefined threshold MC_{thr}, the flow features are forwarded to the CP-IDS via PacketIO.

5.4.3 *Flow and Subflow Scopes in CML-IDS*

The definition of a flow can vary depending on different network management operations [71]. In CML-IDS, a traffic flow is identified using the CRC32 hash function, which considers the 5-tuple packet information, including source and destination IP addresses, source and destination port numbers, and transport layer protocol.

To reduce detection delays caused by waiting to complete the flow, we introduce a sub-flow concept, which consists of the initial N packets of each flow [61]. These initial packets contain sufficient information for training the ML models. In Section 5.8.1, we elaborate on the process of determining the value of N. Based on our

Algorithm 3 : Sub-Flow identification procedure

```
Input : An incoming packet $p$, Sub-flow size $N$
for $p$ do
    $id_j \leftarrow$ CRC32($p_{s_{ip}}, p_{d_{ip}}, p_{s_{port}}, p_{d_{port}}, p_{proto}$);
    $F_j \leftarrow$ Buffer.read($id_j$);
    if $F_j$ is empty then
        Buffer $\leftarrow$ add $F_j$;
    else
        if $len(F_j) < N$ then
            $F_j \leftarrow$ Add features of $p_j$ ;
            $p_j \leftarrow$ Forward ;
        end
        if $len(F_j) == N$ then
            $F_j \leftarrow$ Add features of $p_j$ ;
            $p_j \leftarrow$ Forward ;
            Start classification for the sub-flow $F_j$;
        end
        if $len(F_j) > N$ then
            if Label of $F_j$ == Benign then
                $p_j \leftarrow$ Forward to the appropriate port;
            else
                $p_j \leftarrow$ Drop
            end
        end
    end
end
```

assessment and related work reports (e.g., [23]), we found that utilizing the initial 8 packets led to receiving sufficient information while keeping the detection speed high [61].

After parsing each incoming packet p, the flow ID of p is determined using the CRC32 hash function to collect packets of a flow. The flow buffer retrieves the corresponding flow F based on the computed flow ID. The next processing step is then executed depending on the status of p, as outlined in Algorithm 3. The feature updating and initiation of the classification process for the corresponding flow F depend on the arrival count of packets and whether this count is less than N, which defines the sub-flow or not. To extract the statistical features of a network flow, we utilize NFStream, a Python framework that can extract early statistical and post-mortem flow features [11]. The extracted statistical features are divided into three directions, including source to destination (*src2dst*), destination to source (*dst2src*), and bidirectional (involving packets flowing in both *src2dst* and *dst2src* directions). To determine the direction of each flow, we employ Algorithm 4. Upon the arrival of each packet

Algorithm 4 : Identifying packet direction

```
Input : An incoming packet $p$
(src2dst_no_match, src2dst_empty) $\leftarrow$ $\mathtt{False}$;
$id_j \leftarrow$ CRC32($p_{s_{ip}}, p_{d_{ip}}, p_{s_{port}}, p_{d_{port}}, p_{proto}$);
$F_j \leftarrow$ Buffer.read($id_j$);
if $F_j$ is not empty then
    if $p$.5_tuple $= F_j$.5_tuple then
        $p$ is the src2dst packet of $F_j$;
    else
        src2dst_no_match $\leftarrow$ $\mathtt{True}$;
    end
else
    src2dst_empty $\leftarrow$ $\mathtt{True}$;
end
if src2dst_no_match or src2dst_empty is $\mathtt{True}$ then
    $id_i \leftarrow$ CRC32($p_{s_{ip}}, p_{d_{ip}}, p_{s_{port}}, p_{d_{port}}, p_{proto}$);
    $F_i \leftarrow$ Buffer.read($id_j$);
    if $F_i$ is not empty then
        if $p$.swapped_5_tuple $= F_i$.5_tuple then
            $p$ is the dst2src packet of $F_i$;
        else
            if src2dst_no_match is $\mathtt{True}$ then
                Hash collision happens;
            else if src2dst_empty is $\mathtt{True}$ then
                $p$ results in a new sub-flow $F_j$;
            end
        end
    else
        if src2dst_no_match is $\mathtt{True}$ then
            Hash collision happens;
        else if src2dst_empty is $\mathtt{True}$ then
            $p$ results in a new sub-flow $F_j$;
        end
    end
end
```

p, the features for the corresponding flow are calculated considering the packet's direction. For instance, the creation timestamp of F is set with the timestamp of the first packet arrived for this flow, which is later used to compute the expiration time to terminate packet collection for that flow. Furthermore, all packets contribute to updating bidirectional features.

5.4.4 *DP-IDS Model Confidence Calculation*

The Gini impurity quantifies how often a randomly chosen element from a set would be inaccurately labeled based on the subset's label distribution. In decision trees, it's a common criterion for node splitting [21]. Mathematically, for a set with K classes and p_i denoting the probability of an item being labeled with class i, the Gini impurity is computed as shown in Equation 17:

$$\mathrm{Gini} = 1 - \sum_{i=1}^{K} p_i^2 \tag{17}$$

For binary classification, like distinguishing between attack and benign, the Gini impurity calculation is different, as illustrated in Equation 18:

$$\mathrm{Gini} = 1 - (p_1^2 + p_2^2) \tag{18}$$

This value ranges from 0 to 0.5. A Gini impurity of 0 represents perfect classification (all elements belong to a single class), while 0.5 represents the worst-case scenario (equal probability of belonging to either class).

In decision trees, the goal when splitting a node is to minimize the Gini impurity of child nodes. This means that a split is favorable if it results in child nodes with more homogeneous classes. This iterative process constructs a decision tree by selecting splits that minimize the Gini impurity.

Hence, in classification using a decision tree model, the Gini impurity of leaf nodes (the tree's endpoints) serves as a measure of confidence in the predicted class. Lower Gini impurity values indicate higher confidence, suggesting that the majority of instances at that leaf node belong to a single class.

According to Algorithm 5, the Model Confidence (MC) of the DP-IDS, consisting of an RF model and three DT models, is compared with MC_{thr} to determine whether to forward the flow to the CP-IDS.

Definition — *The MC_{thr} is a predefined threshold value used to determine whether the MC value indicates that the decision from DP-IDS is reliable or if the features should be forwarded to the control plane for further analysis.*

To establish the MC of the DP-IDS, the Gini values of all three DT models are extracted. If all three DTs make consistent decisions, the average of their Gini values is compared with MC_{thr}. If two DTs agree on a decision different from the third one, the average Gini value of the two agreeing DTs is compared with the Gini value of the dissenting DT. Additionally, this average Gini value of the two DTs is compared with MC_{thr}. If any of these comparisons are satisfied, the sub-flow is forwarded to the CP-IDS; otherwise, the decision of the two DTs is accepted for the sub-flow.

Our collaborative flow classification mechanism balances classification accuracy, detection speed, and network latency. Forwarding the sub-flow to the controller yields a more accurate prediction but increases network latency due to transmission time between the programmable switch and controller. Conversely, the DP-IDS predicts a flow with relatively lower accuracy but achieves high detection speed. Thus, MC_{thr} serves as a compromise between sub-flow classification accuracy and latency. Setting MC_{thr} too low results in more sub-flows delivered to the controller, enhancing overall classification accuracy. Conversely, setting MC_{thr} higher leads to more sub-flows classified within the switch using the RFswitch classifier. Therefore, selecting an optimal MC_{thr} is crucial for balancing this trade-off, as discussed in Section 5.8.1.

5.4.5 *Investigating Incompatible Sub-Flow Features*

When deploying an ML model in the programmable data plane, it's essential to account for limitations concerning the flow features on which the ML model can be trained. This is because programmable switches may not be capable of extracting all flow features due to factors such as complex mathematical operations or differences in scale compared to traffic converters like NFStream. Below, we outline the incompatible features and provide explanations for their removal from the training of the ML model.

Architecture-derived Features:

Algorithm 5 : Decision making process using Gini impurity

```
Input : Sub-flow $F$, RF model contains of three DTs $DT_i, DT_j, DT_l$ with the
        final prediction of $C_i, C_j, C_l$, Model Confidence threshold $MC_{thr}$
Output : Decision about forwarding the sub-flow $F$
1  if $C_i = C_j = C_l$ then
2      $\bar{G}_{i,j,l}(F) = \frac{1}{3}(G_i(F) + G_j(F) + G_l(F))$;
3      if $\bar{G}_{i,j,l}(F) \geqslant MC_{thr}$ then
4          Forward $F$ to CP-IDS;
5      else
6          Accept the detection result;
7      end
8  else
9      if $C_i = C_j and C_i \neq C_l$ then
10         $\bar{G}_{i,j}(F) = \frac{1}{2}(G_i(F) + G_j(F))$;
11         if $\bar{G}_{i,j}(F) \geqslant G_l(F)$ or $\bar{G}_{i,j}(F) \geqslant MC_{thr}$ then
12             Forward $F$ to CP-IDS;
13         else
14             Accept the detection result;
15         end
16     end
17 end
```

Table 11: Comparison of Packet Inter Arrival Time (PIAT) measurements between programmable switch (BMv2) and network traffic converter, NFStream. The results indicate the differences in time measurement, which can affect training of the ML model.

Time-Related Features	**Measured in BMv2 (μs)**	**Measured by NFStream (μs)**
Min. PIAT (*dst2src*)	3410.7	2000
Min. PIAT (*bidirectional*)	255.6	0
PIAT (between 1st and 2nd packet)	729.4	0

In order to ensure the broad applicability of CML-IDS across all network traffic passing through a switch, it is imperative to exclude features derived from network architecture (such as IP addresses, Mac addresses, and port numbers).

Temporal Features: These features are calculated within the DP-IDS by accessing the *ingress_global_timestamp* and *egress_global_timestamp* values of the P4 language, which indicate the times when a packet enters and exits the switch, respectively. These timestamps are expressed in microseconds within the programmable switch (BMv2), while NFStream measures temporal features in milliseconds. To ensure uniformity in time units, the results obtained by NFStream are multiplied by 1000. However, a notable discrepancy persists due to NFStream's lower precision in time measurement. Specifically, NFStream assigns a value of 0 to temporal features that are less than 1 millisecond. Table 11 illustrates the disparity between the measured values of temporal features in BMv2 and NFStream. The BMv2 measurements represent the average of ten individual readings. Furthermore, it demonstrates that NFStream's temporal features are less accurate compared to BMv2. Consequently, we have excluded temporal features from the training dataset.

Complex Computational Features: The constraints of the P4 language present a hurdle for computing intricate features that entail loops or divisions (such as determining packet size standard deviation) within the DP-IDS. Consequently, it is currently impractical to calculate most of these features within the programmable switch.

To address this challenge, [23] proposed employing the Exponential Weighted Moving Average (EWMA) method to estimate the mean value of a feature instead of computing the exact mean. This method can be utilized within the switch for estimation purposes. For instance, the estimation of the *src2dst_mean_ps* feature (where *ps* denotes packet size) is conducted using Equation 19, employing a smoothing factor of 0.5.

$$\bar{PS}^{t}_{src2dst} = \frac{\bar{PS}^{t-1}_{src2dst} + Payload^{t}_{src2dst}}{2} \tag{19}$$

Here, $\bar{PS}^{t}_{src2dst}$ represents the current mean packet size, and $\bar{PS}^{t-1}_{src2dst}$ denotes the previous mean packet size. As the P4 language does not support the division operation in Equation 19, the division by 2 is achieved by right-shifting the binary value by one position. However, the accuracy of the estimated mean value using EWMA may be compromised if the payloads of packets within a sub-flow vary significantly [61]. In our implementation, given that the maximum number of packets per sub-flow remains constant (i.e., 8), the *bidirectional_mean_ps* feature value can be accurately computed by right-shifting the binary value of the *bidirectional_bytes* feature (which represents the sum of packet sizes of extracted packets within a sub-flow) by three positions. Consequently, all features related to mean values, except for *bidirectional_mean_ps*, are excluded from the training dataset. In fact, the number of src2dst or dst2src packets of a sub-flow (i.e., 8 packets) can vary depending on the traffic pattern; however, the number of bidirectional packets always remains 8 due to the definition of subflow. Furthermore, computing features associated with standard deviation are currently unattainable within the existing P4 switch pipeline architecture.

The total number of flow features that can be extracted after excluding incompatible features is 59. Among them, 20 features are selected as the most important features leveraging the EFS method explained in Chapter 3. The list of 20 selected features is available in Section A.4. Reducing the number of features can also reduce the required computational resources for training the ML model in the DP-IDS. Additionally, it can reduce the risk of overfitting the mode to a special network traffic pattern [84].

5.5 CML-IDS IMPLEMENTATION

In this section, we provide a detailed explanation of the implementation of the proposed CML-IDS. To enhance clarity, we outline the implementation of each defined module in the CP-IDS (as described in Section 5.4.1), DP-IDS (as outlined in Section 5.4.2), and those required for communication between them.

5.5.1 *CP-IDS Implementation*

The control plane is developed using two modules utilizing the Python programming language. These modules include the Machine Learning Module and the P4 Program Generator. In the following each of them is explained.

Implementation of Machine Learning Module

The Machine Learning Module within the control plane is responsible for a pre-processing pipeline, which involves aggregating the packet-based dataset into a subflow-based dataset. It also handles the creation of training datasets for building each ML classifier, conducting hyperparameter searches, and performing feature selection. Additionally, this module is responsible for training and constructing the embedded DP-IDS within the programmable switch and each ML model within the ensemble CP-IDS. Furthermore, it is tasked with classifying the received sub-flows that couldn't be classified by the DP-IDS. In the following, the implementation of each of these sub-modules are explained.

Converting Network Traffic Dataset:
To train the ML models and evaluate the CML-IDS, we primarily utilize the CICIDS2017 dataset [150], which comprises both benign and attack traffic. The attacks mainly consist of BruteForce, DoS/DDoS, and Botnet attacks. The dataset provides two distinct types of extracted traffic data: PCAP files containing raw packet-based traffic and CSV files containing flow-based datasets with traffic labels for each flow entry, converted using their designated traffic converter.

In order to ensure the adaptability of the CML-IDS to any type of network traffic dataset, we designed a preprocessing pipeline that functions uniformly across all datasets. Consequently, we opt to use the raw packet-based datasets, providing greater flexibility and generalization in processing and training classifiers. To facilitate this process, we employ the NFStream flow aggregation tool [11] to convert the packet-based datasets into flow-based datasets and extract customized subflow-based features. These features encompass core features, post-mortem statistical features, and early statistical features, including direction features, packet size, and inter-arrival time for the first N packets within a flow (F).

NFStream offers configurable attributes for extracting flow features, each serving specific purposes:

- accounting_mode: When set to 3, it accumulates payload lengths solely for byte-related features.
- udps: This setting employs our customized FlowSlicer class, facilitating the extraction of subflow-based features from the first N packets within a flow.
- statistical_analysis: It activates the extraction of post-mortem flow statistical features.
- splt_analysis: Set to N, it stores early statistical features for the first N packets within F.

Our customized FlowSlicer copies core and post-mortem statistical features during packet aggregation into flows. The process concludes either when bidirectional packets reach the maximum number of N (based on the sub-flow definition) or when NFStream ends flow aggregation due to timeout conditions. Subsequently, subflow-based features were created to contain information from initial packets. Entries are categorized based on bidirectional packet count:

- If bidirectional packets are less than N, the entry is discarded from the flow-based dataset.
- If bidirectional packets equal N, the entry is retained.

The extracted features are utilized to train CP-IDS and DP-IDS, as well as evaluate their performance.

Dataset Preprocessing:

NFStream initially sets the early statistical features to -1 to denote the absence of a packet. However, managing negative values in the P4 language is complex. Therefore, a workaround is implemented where all early statistical features are incremented by 1 to prevent negative values. Considering the first packet of a given flow F always serves as the src2dst packet, only the directional feature *splt_direction_1_1* remains for the first packet, while subsequent packets' directional features exclude those with the suffix 0. Following the concatenation of all datasets, data points undergo shuffling to enhance generalization. The final selection of features is based on their compatibility with the P4 switch, detailed in Section 5.4.5.

Reading and Writing Flow Features:

Our flow storage strategy implementation utilizes a single register as the buffer to store all extracted flow entries. This compact approach significantly reduces the need for registers and enables the storage of a wide range of flow features. All compatible features of a flow, along with the flow metadata, are serialized into a bit string using the bit concatenation operation supported by the P4 language [61]. This bit string is then stored in the flow buffer, with the index determined by the flow ID computed using the CRC32 hash function.

The process of accessing a flow entry from the flow buffer begins by providing a specific sub-flow ID to the flow buffer. Next, the sub-flow bit string is retrieved from the flow buffer and subsequently converted into a flow structure data type similar to the struct in the C programming language. Each feature value is then assigned to a corresponding feature parameter.

Tuning Hyperparameters of ML models:

The final proper network traffic datasets are split into training, validation, and test datasets with the proportion of 70%, 10%, and 20%, respectively. The DP-IDS is an RF model with three DTs tailored to the programmable switch's constrained computational resources. To ascertain the optimal hyperparameter of maximum depth for each DT, we leverage the grid search cross-validation technique, which is employed in the training and validation set. This involves calculating the macro-average F1 scores for all RF classifiers with three DTs, varying the maximum depths from 1 to 10. The objective is to tune the hyperparameters to achieve a high detection performance with fewer DT models in the RF model. Figure 19 depicts the RF model performances with various maximum depth values. According to the result, a maximum depth of 5 yielded a macro-average F1 score exceeding 0.9424. Consequently, this value is selected as the optimal maximum depth for each DT within the RF model. Subsequently, the trained RF model is passed to the P4 Program Generator module to generate the ML-related P4 code snippets.

In contrast to the data plane, the control plane exhibits a higher level of flexibility and programmability, enabling the deployment of more advanced ML models. The CP-IDS deployed in the control plane consists of three distinct ML models: an MLP classifier, an RF classifier, and an XGBoost classifier. Further details about these

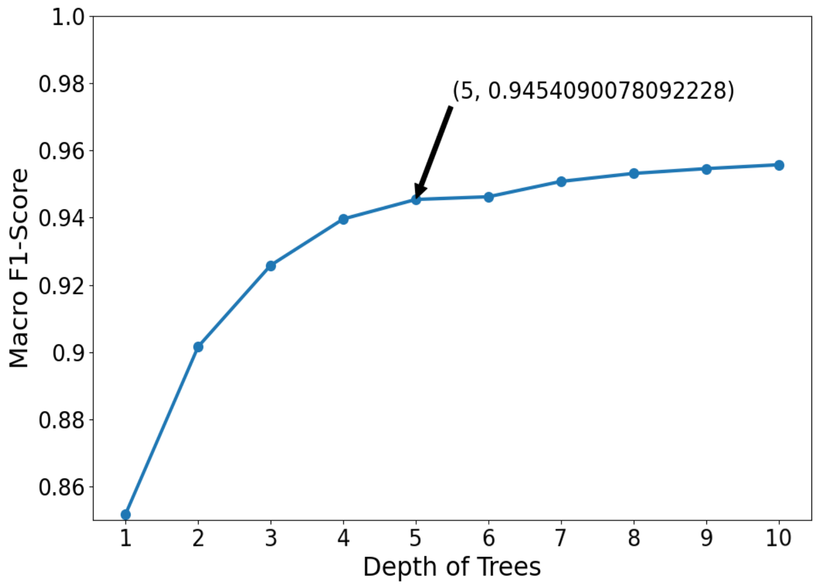

Figure 19: Results of grid-search cross-validation for determining the optimized value for the "maximum depth of DT" hyper-parameter. The F1-Score shows a slight improvement after a depth of 5.

Table 12: Selected hyperparameters for the ML models deployed in the DP-IDS and the CP-IDS.

Parameters	**Values**
CP-IDS:	
XGB	max_depth: 25, tree_method: 'approx', scale_pos_weight: 40
RF	n_estimators: 500, min_samples_leaf: 5, max_depth: 10
MLP	number of hidden layers: 14, batch_size:2048, epochs:200
DP-IDS:	
RF	n_estimators: 3, min_samples_leaf: 1, max_depth: 5

models can be found in Section 2.4.1. The ensemble ML classifier generates the final prediction by aggregating the individual predictions produced by each of these three classifiers. All compatible flow-based features (totaling 59 features) obtained in Section 5.4.5 are utilized for training, evaluation, and test datasets. Table 14 presents the hyperparameters of both the DP-IDS and the CP-IDS.

Final Decision of the CP-IDS:

According to Algorithm 6, upon receiving a sub-flow F that couldn't be classified at the data plane, each classifier within the ensemble CP-IDS evaluates the sub-flow and generates probabilities indicating whether it belongs to the benign or attack class. The final probability, used for the final decision, is computed by averaging each probability. If the final probability for a benign flow exceeds that for an attack, the ensemble ML classifier assigns the final predicted label $C_{CP}(F)$ for the received flow as 0, indicating the sub-flow F as benign. Conversely, if $C_{CP}(F)$ is set to 1, it indicates that the sub-flow F is classified as an attack.

Implementation of P4 Program Generator Module

A P4 program encompasses various stages to manage incoming packets, including header processing, parsing, checksum verification, and ingress processing. Among these stages, ingress processing is pivotal as it orchestrates the MA pipeline to handle parsed packets. As discussed in subsection 5.2.2, the RF classifier trained by the machine learning module is represented through MA tables and their entries within the pipeline. Hence, dynamically adapting the P4 program becomes essential to accommodate changes in DP-IDS.

The automatically generated components of the P4 program are listed below. More details about them can be found in Section 5.2.2.

- MA tables to represent each tree level of the RF model deployed in the DP-IDS.
- Table entries to depict roots, intermediate nodes, and leaves within each tree of the DP-IDS.
- Classification logic to predict the sub-flow label through the MA pipeline.
- Feature comparison logic to construct the classification path.
- Conversion of struct data types into bit strings to store sub-flow entries in the flow buffer.
- Conversion of bit strings into struct data types to retrieve sub-flow entries from the flow buffer.

Algorithm 6 : Ensemble ML classifier flow prediction algorithm.

```
Input : Sub-flow $F$
Output : $C_{CP}(F)$: predicted class by the ensemble ML model.
1  $[Pr_{benign1}, Pr_{attack1}] \leftarrow$ MLP.predict($F$) ;
2  $[Pr_{benign2}, Pr_{attack2}] \leftarrow RF_{CP-IDS}$.predict($F$) ;
3  $[Pr_{benign3}, Pr_{attack3}] \leftarrow$ XGBoost.predict($F$);
4  $Pr_{benign} = \frac{1}{3}(Pr_{benign1} + Pr_{benign2} + Pr_{benign3})$;
5  $Pr_{attack} = \frac{1}{3}(Pr_{attack1} + Pr_{attack2} + Pr_{attack3})$;
6  if $Pr_{benign} > Pr_{attack}$ then
7      $C_{CP}(F) \leftarrow 0$ ;
8  else
9      $C_{CP}(F) \leftarrow 1$;
10 end
```

Subsequently, the entire P4 program is compiled into JSON and P4Info files using the p4c[1] compiler. These compiled files encapsulate crucial information about the P4 program and the metadata of its entities. These files are then dispatched to the Data Plane Control Module to initialize the BMv2 switch.

p4c Compiler:
The p4c compiler is an essential tool for converting P4 program source code into JSON and P4Info files, which are crucial for establishing connectivity between the controller and BMv2 switch. The JSON file contains data structures, MA tables, and packet processing logic from the P4 program, while the P4Info file provides descriptions and metadata for tables, counters, registers, PacketIn, and PacketOut headers. The table metadata helps the controller understand each table's structure in the switch, allowing it to populate corresponding table entries accurately. Descriptions of PacketIn and PacketOut headers aid in parsing incoming packets to extract flow feature values and set header fields in outgoing packets. Additionally, p4c supports generating visual representations of P4 programs, making it easier to visualize the workflow within the switch.

5.5.2 *DP-IDS Implementation*

To implement CML-IDS, the software switch BMv2 [2] is utilized within the data plane. It handles incoming network packets and conducts preliminary subflow-based traffic classification using DP-IDS. This switch operates according to a P4 program programmed in the P4 language.

Behavioral Model Version 2 (BMv2)

BMv2, the second iteration of the reference P4 software programmable switch supported by the p4 language project, operates on the P4 language and serves as a platform for developing and testing P4 programs in the data plane.

To enable efficient packet reception and forwarding, the BMv2 software switch requires connections to network interfaces. In our testing environment, we create two virtual network interfaces using the IP command on the Linux machine. Each interface's Maximum Transmission Unit (MTU) is set to the maximum value of 65535 bytes, ensuring seamless processing of all packets within the switch.

The trained DP-IDS from the machine learning module in the control plane is deployed within BMv2 as MA tables and entries, functioning as an in-network intrusion detection system.

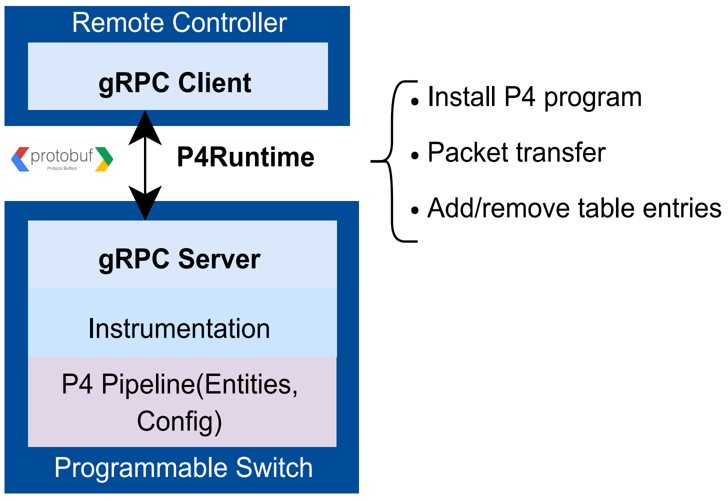

Figure 20: DP-IDS and CP-IDS interact through P4Runtime API, and the data between them is serialized using protobuf.

5.5.3 *Collaboration between DP-IDS and CP-IDS*

The interaction between the data plane and control plane is enabled by the P4Runtime API [64], utilizing a gRPC connection between the switch and the controller, as illustrated in Figure 20. This connection employs Protocol Buffers (Protocol Buffer3) to serialize the data format for communication between the two planes. Collaborative flow classification is facilitated by integrating the DP-IDS in the BMv2 switch, the CP-IDS in the control plane, and the connection established through the P4Runtime API.

P4Runtime API

This API supports various functions for managing the BMv2 switch by the controller. Specific use cases in our implementation are detailed in the Interaction between the Switch and Controller. It involves the P4Runtime Shell[3] as the P4Runtime client and the P4c compiler, which is explained in Section 10.

P4Runtime Shell
The P4Runtime client is customized and deployed within the control plane to facilitate communication with the P4Runtime server situated in the data plane. Various implementations of the P4Runtime client exist, including p4runtime_lib[4] and P4Runtime Shell. Both clients necessitate the JSON and P4Info files compiled from the P4 program to establish connectivity with the BMv2 switch. Establishing the

1 https://github.com/p4lang/p4c
2 https://github.com/p4lang/behavioral-model
3 https://github.com/p4lang/p4runtime-shell
4 https://github.com/p4lang/tutorials/tree/master/utils/p4runtime_lib

connection requires specifying the gRPC socket address, which is set during BMv2 switch initialization via the simple_switch_grpc CLI. However, PacketIO support is lacking in p4runtime_lib, hindering packet transfer between the data plane and the control plane. Consequently, our approach employs P4Runtime Shell as the client.

Within the P4Runtime Shell, the PacketIn handler enables packet capture from the designated CPU port, defined during BMv2 switch initialization. Captured packets undergo parsing and processing within the controller. Additionally, the PacketOut handler in P4Runtime Shell allows specifying header field values in outgoing packets, which are then sent to the switch. In our setup, incoming packets convey flow information, while outgoing packets encapsulate the predicted flow label.

Protocol Buffers (protobuf)

Protocol Buffers, developed by Google, is a language-agnostic data serialization format designed to efficiently encode structured data into a compact binary format. Its versatility makes it invaluable for transmitting data over networks or storing it in files. In the context of the P4Runtime API, Protocol Buffers plays a crucial role in defining messages and semantics that govern the interface between the switch and controller. The P4Runtime API enables the controller to access P4 entities declared in the P4Info metadata, which is outlined within the P4Info protobuf file. This metadata provides comprehensive information about MA tables, packet header fields exchanged between the controller and switch, and other data structures declared within the P4 program. By leveraging Protocol Buffers, the P4Runtime API establishes a standardized and efficient means for the controller to traverse these entities. This allows for the extraction of header fields from PacketIn and PacketOut packets, which contain feature values of uncertain flows and the ultimately predicted flow label, respectively.

Remote Procedure Call (RPC)

gRPC, an open-source framework developed by Google, provides a high-performance RPC (Remote Procedure Call) solution that enables seamless communication between network entities. It operates using the Protocol Buffers data serialization format, facilitating efficient data exchange.

As illustrated in Figure 20, within the scope of the P4Runtime API, gRPC functions as the foundational element for establishing connectivity between the P4Runtime server, located within the switch, and the P4Runtime client deployed in the control plane. This robust connection mechanism guarantees seamless and dependable communication between the switch and the controller.

In the data plane, the software switch BMv2 is launched using either `simple_switch`[5] or `simple_switch_grpc`[6] CLI. The `simple_switch` CLI required the JSON file compiled from the P4 program to start the BMv2 switch. Once launched, the BMv2 switch is equipped with the installed P4 program. However, the `simple_switch` CLI

5 https://github.com/p4lang/behavioral-model/tree/main/targets/simple_switch
6 https://github.com/p4lang/behavioral-model/tree/main/targets/simple_switch_grpc

lacks the capability to connect to the control plane via the P4Runtime API. To address this limitation, the alternative version `simple_switch_grpc` CLI is utilized in our case. `simple_switch_grpc` supports the P4Runtime API, enabling the establishment of a gRPC connection to the control plane. The invocation of `simple_switch_grpc` is as follows:

```
simple_switch_grpc - -no-p4 \
     -i <PORT1>@<IFACE1> -i <PORT1>@<IFACE1> \
     - -grpc-server-addr <IP>:<TCP PORT> -cpu-port <CPU PORT>
```

The option `-no-p4` allows for initiating the BMv2 switch without utilizing a JSON file, meaning the switch will start without a pre-installed P4 program. Instead, the P4 program will be installed from the control plane via the P4Runtime API. The flag `-i <PORT>@<IFACE>` sets the port number for each network interface. Using `-grpc-server-addr <IP>:<TCP PORT>` provides a socket address to run the P4Runtime server, enabling it to receive connections from P4Runtime clients. Finally, `-cpu-port <CPU PORT>` specifies a CPU port number and activates the PacketIO support of the P4Runtime API. Packets sent to this CPU port number will be forwarded to the P4Runtime clients in the control plane for ensemble ML classification.

5.6 EVALUATION RESULTS

In this section, we evaluate the proposed CML-IDS using network traffic sourced from the CICIDS2017 datasets. We assess its detection performance, speed, and network load utilization. To compare the effectiveness of the CML-IDS, we consider baselines, including in-network NIDS, which deploys ML models solely in the data plane, and the ML model deployed solely in the control plane. In our evaluation, we investigate how adjustments in the MC_{thr} parameter impact detection performance and network load, with the goal of finding a threshold that optimizes both factors.

Subsequent sections provide detailed information on the evaluation metrics and setup, including hardware and software environments. Each evaluation scenario is then presented alongside the corresponding results.

5.6.1 *Evaluation Metrics*

To evaluate the detection performance of CML-IDS, We utilize the macro-average of precision, recall, and F1 Score. The F1 Score is a reliable evaluation metric as it accounts for both false positives and false negatives, thus providing a more precise evaluation of detection performance. In our analysis, the positive class denotes attack traffic, while the negative class signifies benign traffic. Given that benign flows usually dominate network traffic, we chose to compute the macro-average of the F1 Score. This ensures fair treatment of each class, irrespective of its occurrence frequency or any imbalances in the dataset [155]. The computation formulas for these metrics are described as follows.

False Negative (FN)
FN arises when the ground truth label is positive, indicating an attack flow, yet the model incorrectly detects it as belonging to the negative class or benign flow.

False Positive (FP)
FP arises when the ground truth label is negative, indicating a benign flow, yet the model incorrectly detects it as belonging to the negative class or attack flow.

True Negative (TN)
TN occurs when the ground truth label is negative, indicating a benign flow, and the model correctly detects it as belonging to the negative class or benign flow.

True Positive (TP)
TN occurs when the ground truth label is positive, indicating an attack flow, and the model correctly detects it as belonging to the positive class or attack flow.

Precision
Precision assesses the accuracy of positive predictions by measuring the ratio of true positive cases among all instances predicted as positive, indicating the model's ability to identify relevant instances accurately [140]. The formula to compute precision is expressed in Equation 20.

$$Pr = \frac{TP}{TP + FP} \tag{20}$$

where Pr represents precision.

Recall
Recall (Re), known as sensitivity, measures the ability of an ML model to correctly identify all relevant instances from the dataset by calculating the ratio of true positive cases to the sum of true positive and false negative cases [140]. The formula to compute recall is expressed in Equation 21.

$$Re = \frac{TP}{TP + FN} \tag{21}$$

where Re represents recall.

F1 Score
The F1 score offers a comprehensive assessment of the model's performance by incorporating both precision and recall. A high F1 score indicates that the model achieves high precision and recall, reflecting strong classification performance [140]. The F1 score is computed using the formula shown in Equation 22.

$$F1 = \frac{2 \times Pr \times Re}{Pr + Re} \tag{22}$$

where F1 represents F1 score.

Macro Average Metrics
The formulas to compute each macro-average F1 Score metric in our evaluation are expressed in Equation 23.

$$F1_{macro-avg} = \frac{F1(Benign) + F1(Attack)}{2} \tag{23}$$

5.7 EVALUATION SETUP

In this section, we outline the specifics of our evaluation setup, which include the hardware and software environment settings. We clarify the parameter setup for our evaluation in Section 5.7.2. Furthermore, details of the ML models' hyperparameters are available in Table 14.

5.7.1 *Evaluation Environment*

We assessed the performance of our proposed CML-IDS using a virtual machine operating on the Ubuntu system. The hardware specifications of this virtual machine are provided in Table 13. For the data plane, we employed a BMv2 software switch as the programmable switch, deployed alongside the controller on the same machine and locally connected to the BMv2 switch.

For training and evaluation, we utilize datasets from CIC-IDS2017 containing Brute Force, DoS, DDoS, and Botnet attacks. Each dataset is replayed at its original speed using Tcpreplay[7] to replicate real-world traffic forwarding conditions.

Table 13: Specifications of the hardware used for conducting CML-IDS.

Component	**Specifications**
Operating System	Ubuntu 20.04.2
CPU	4 Cores, Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz
RAM	126 GB

5.7.2 *Evaluation Parameters*

Table 14 enumerates the parameter configurations employed in our assessment. The length of the bit string represents the memory usage needed for storing a single flow entry, calculated by summing the lengths of all flow features. In our case, the

7 https://tcpreplay.appneta.com/

Table 14: CML-IDS flow-based parameters.

Parameters	**Values**
Number of Registers	1
Length of bit string	1225 bits
Size of flow buffer	10,000,000
Expiration timeout	30 min
MC_{thr}	[0.1, 0.2, 0.3, 0.4]

combined length of all features equals 1225 bits. The size of the flow buffer determines the maximum number of simultaneous flow entries it can accommodate, set at 10,000,000 to effectively handle hash collisions. The expiration timeout dictates the maximum duration a flow entry remains in the buffer; entries exceeding this timeout are discarded. NFStream typically aggregates packets into flows with a default active timeout of 30 minutes, which we maintain for consistency in our evaluation.

In our assessment, the MC_{thr} plays a crucial role in determining the confidence level of the DP-IDS within the programmable switch. Similar to the Gini impurity in concept, MC_{thr} defines the reliability of DP-IDS classifications. When a flow's detection confidence falls below MC_{thr} (with a smaller Gini value indicating higher confidence), the DP-IDS considers it as a reliable detection and utilizes it as the final classification for that flow entry. Conversely, flows with detection confidences exceeding MC_{thr} are forwarded to the CP-IDS and use its detection as the final result. A lower MC_{thr} imposes a stricter criterion for accepting classification within the switch, leading to more flows being forwarded to the controller. MC_{thr} values range between 0 and 0.5. To ensure the stability of our evaluation results, each threshold has been evaluated three times.

5.8 EVALUATION RESULTS AND ANALYSIS

In this section, we analyze the detection performance of CML-IDS from various perspectives and compare it with different baselines. This comparison aims to elucidate our ultimate objective of enhancing detection performance and detection speed while minimizing network load.

5.8.1 *Analysis of the Optimized Number of Packets in a Sub-flow*

Our main objective, as described in section Section 5.4.4, is to quickly classify each flow. To accomplish this goal, we rely on DP-IDS, which has the ability to categorize flows at line speed. Furthermore, we condense the total number of packets within a flow to a set of N initial packets, defining it as a sub-flow. This approach eliminates the necessity to wait for flow completion.

In this section, we explore the effectiveness of different numbers of initial packets in providing valuable insights for distinguishing between attacks and benign flows.

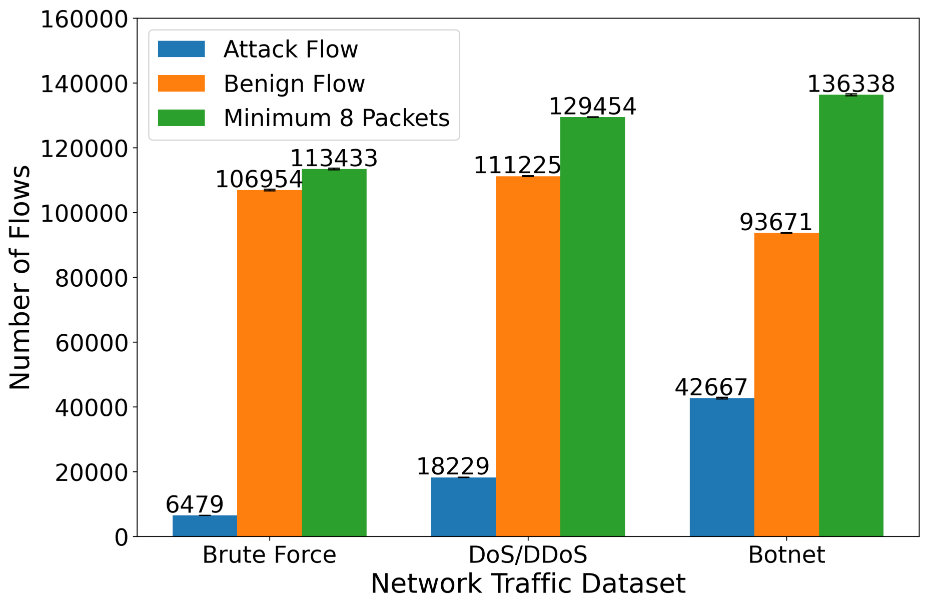

Figure 21: Distribution of benign and attack flows within each network traffic dataset. Flows with more than eight packets are annotated in CML-IDS, which considers eight initial packets as a sub-flow.

Specifically, we examine quantities of 4, 8, 15, and 30 initial packets for this analysis. Models are trained using datasets extracted from each of these packet quantities, and their performance is evaluated using a validation dataset. Importantly, during this phase of determining the optimal number of packets, the evaluation is limited to the training and validation datasets.

The findings indicate a substantial reduction in both FP and FN sub-flows, averaging 62.9% when transitioning from 4 to 8 initial packets. This shift correlates with an overall enhancement in detection performance.

As the number of initial packets increases from 8 to 15 and then to 30, the improvement observed in the process diminishes to 7% and 13%, respectively. This marginal increase compared to the difference between 4 and 8 initial packets suggests that a sub-flow size of 8 packets should be selected in order to minimize delays in the detection process initiation and the need for additional packet waiting.

5.8.2 *Analyzing Flow Distribution Extracted in Programmable Switch*

To understand the network traffic dataset used in evaluating our proposed CML-IDS, we initially annotate sub-flows for further analysis and investigate the distribution of attack and benign flows within the dataset. Annotating of sub-flows is done leveraging victim and attacker IP addresses from the CICIDS2017 dataset (details in Appendix A.2). The ground-truth labels are utilized by the programmable switch to compute FN, FP, TN, and TP alongside the predicted labels.

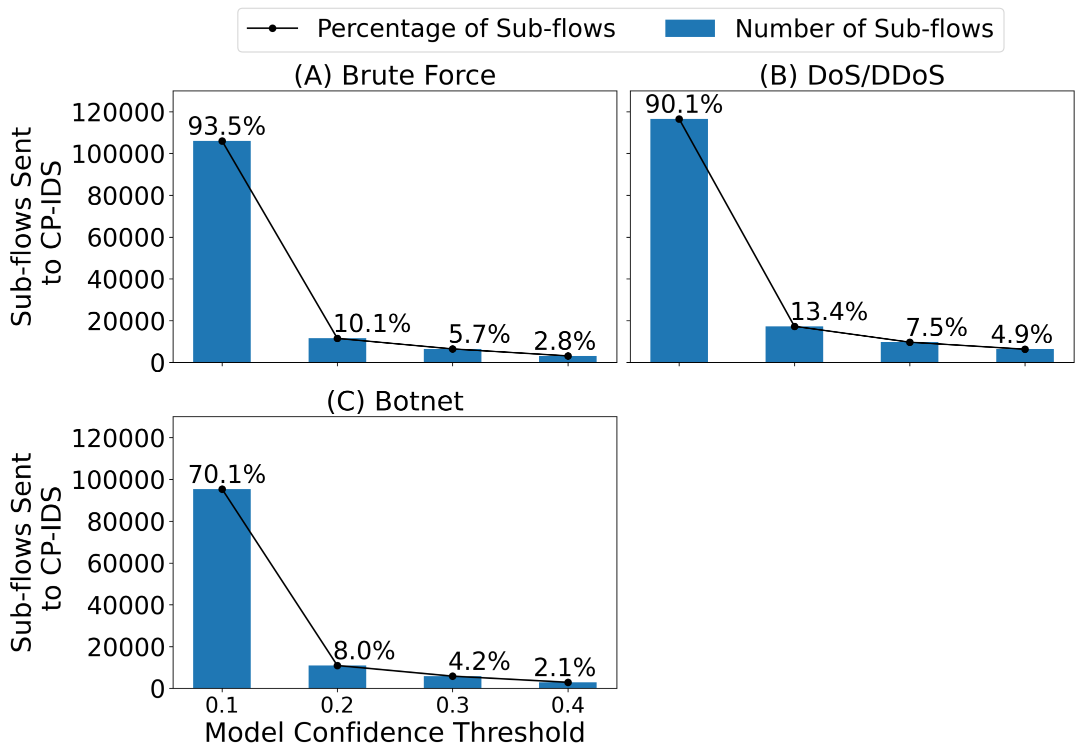

Figure 22: Number of sub-flows forwarded to the control plane and classified by the CP-IDS based on different model confidence thresholds. Furthermore, the percentage of these forwarded sub-flows is indicated in each figure. *The similar figure is in [61].*

In Figure 21, the distribution of benign and attack flows in each dataset, categorized by attack type, is presented. The values shown above each bar represent the averages from 12 evaluation runs (each with the model confidence threshold $\mathrm{MC_{thr}}$ evaluated three times), with error bars indicating the standard deviation. Minimal standard deviation signifies the consistent flow extraction capability of the programmable switch.

5.8.3 *Impact of $\mathrm{MC_{thr}}$ on the Percentage of Forwarded Sub-flows*

This section examines how different values of $\mathrm{MC_{thr}}$ can influence the percentage of sub-flows forwarded to the control plane for classification using CP-IDS.

As depicted in Figure 22, for $\mathrm{MC_{thr}} = 0.1$, over 90% of sub-flows from the Brute-Force and DoS/DDoS network traffic datasets, and more than 70% of sub-flows from the Botnet dataset, are forwarded to the CP-IDS. This high percentage is due to the requirement of a very high confidence level from the DP-IDS to refrain from forwarding the sub-flow to the CP-IDS. However, as expected, with an increase in $\mathrm{MC_{thr}}$, the proportion of sub-flows forwarded to the CP-IDS decreases, indicating an increased tolerance for accepting the detection results from the DP-IDS.

5.8.4 *Detection Performance of CML-IDS*

This section evaluates the overall detection performance achieved through the collaboration of the DP-IDS and the CP-IDS using various MC_{thr}. Additionally, the results are compared with a "baseline," which refers to the scenario where the CP-IDS in the control plane is inactive, and the final prediction (Attack/Benign) for all sub-flows is determined solely by the DP-IDS. To clarify the distinctions, we also measure the misclassification rate of CML-IDS using Equation 24.

$$MR = \frac{(FP + FN)}{(FP + FN + TP + TN)} \tag{24}$$

Table 15 illustrates the impact of different MC_{thr} values on the detection performance and misclassification rate. The results indicate that the macro-average F1 Score improves when utilizing CML-IDS (for all MC_{thr} values) compared to the baseline approach, which relies solely on the DP-IDS. Moreover, the results of the misclassification rate demonstrate that CML-IDS effectively reduces the percentage of misclassifications. This improvement is attributed to the CP-IDS being more sophisticated than the DP-IDS; therefore, forwarding sub-flows that are challenging for the DP-IDS to detect (based on its confidence) to the CP-IDS can enhance the overall detection performance.

In terms of the different model confidence thresholds, MC_{thr} of 0.1 yields the best classification performance, as it causes to forward a large number of sub-flows to the control plane for classification by the CP-IDS. As shown in Figure 22, more than 90% of the classified sub-flows for Brute Force and DoS/DDoS attacks, as well as 70% of Botnet attacks, are classified within the controller. On the other hand, MC_{thr} of 0.4 results in relatively the lowest detection performance among the collaborative classification since most sub-flows are classified within the DP-IDS. The frequent packet transfer between the programmable switch and controller could overwhelm the network bandwidth between the data plane and the control plane. This drawback is not aligned with our design goal of detecting flows as early as possible.

Table 15: Relative detection performance (DP) and misclassification rate (MR) of different network traffic datasets for different MC_{thr}. After conducting three individual experiments, the standard deviations for all values are below 0.001. *The table is extracted from [61].*

MC_{thr}	BruteForce		DoS/DDoS		BotNet	
	DP(%)	MR(%)	DP(%)	MR(%)	DP(%)	MR(%)
0.1	96.8	0.7	83.0	7.1	95.5	3.7
0.2	96.7	0.7	83.7	7.2	96.6	2.9
0.3	***96.7***	***0.7***	***87.0***	***6.3***	***96.7***	***2.6***
0.4	88.1	3.1	82.0	7.9	94.7	4.4
Baseline	***84.5***	***4.4***	***81.2***	***8.9***	***93.1***	***5.7***

Table 16: Comparison of detection time between DP-IDS and CP-IDS for different MC_{thr} values. More sub-flows are forwarded to CP-IDS for low MC_{thr}, resulting in increased detection time. *The table is extracted from [61].*

Model Confidence Threshold	**Detection Time in the DP-IDS (second)**	**Detection Time in the CP-IDS (second)**
0.1	0.067 (± 0.0)	137.066 (± 61.417)
0.2	0.075 (± 0.023)	0.546 (± 0.05)
0.3	0.057 (± 0.001)	0.361 (± 0.04)
0.4	0.057 (± 0.001)	0.354 (± 0.044)

The maximum difference is for $MC_{thr} = 0.3$, displayed in bold in Table Table 15. At this threshold, the detection performances for BruteForce, DoS/DDoS, and Botnet attacks are improved by 12.2%, 5.8%, 3.6%, respectively, and the misclassification rates are reduced by 84.0%, 29.2%, 54.3%, compared to the baseline.

5.8.5 *Impact of* MC_{thr} *on Detection Time*

In the domain of network security, promptly identifying attack patterns is crucial for minimizing potential harm and maintaining the integrity of the network. Depending on the network task, the delay tolerance can be different [181]. This section explores the investigation into the time taken to detect attacks for different MC_{thr} values, while also comparing the detection speeds of DP-IDS and CP-IDS. The impact of directing sub-flows to the CP-IDS on the overall detection time is depicted in Table 16. Notably, when $MC_{thr} = 0.1$ is employed, the detection time rises due to a higher proportion (93.5%, 90.1%, and 70.1% for various attacks as shown in Figure 22) of sub-flows being routed to the CP-IDS. Consequently, the classification of sub-flows using the CP-IDS for $MC_{thr} = 0.1$ demands more time. The significant delay in detection observed with the CP-IDS for $MC_{thr} = 0.1$ emphasizes the drawbacks of solely depending on ML models deployed in the control plane for effective intrusion detection. However, the detection time for other MC_{thr} values remains below 1 second, indicating swift detection.

5.8.6 *Selecting the Optimal* MC_{thr}

To strike a balance between achieving high detection performance and maintaining low network load, various metrics are taken into account when determining the suitable MC_{thr} value. While Table 15 illustrates strong detection performance across MC_{thr} values of 0.1, 0.2, and 0.3, Figure 22 indicates that a higher number of sub-flows are routed to the CP-IDS for $MC_{thr} = 0.1$ and $MC_{thr} = 0.2$ compared to $MC_{thr} = 0.3$, leading to increased network latency. Therefore, selecting $MC_{thr} = 0.3$ is more aligned with the primary objectives of CML-IDS.

In addition to the evaluation metrics mentioned earlier, the distribution of various types of classified sub-flows forwarded to the controller is monitored to evaluate the detection performance of CML-IDS. These sub-flows, initially classified by the DP-IDS with confidence levels below the MC_{thr}, are categorized based on their classifications within the DP-IDS, namely FN, FP, TN, and TP. To improve classification efficiency, sub-flows classified as FN or FP should be prioritized for forwarding to the controller since erroneous predictions made within the switch can be corrected by the CP-IDS. However, it's essential to consider the total number of sub-flows forwarded to the CP-IDS to prevent an increase in network bandwidth load. Conversely, sub-flows correctly classified within the switch as TN or TP should be minimized from being forwarded to the controller to decrease network bandwidth usage and reduce controller workload. Therefore, the objective is to increase the proportion of flows classified as FN or FP among all flows sent to the controller.

The findings depicted in Figure 23 reveal that across all network traffic datasets, the proportion of FP and FN sub-flows forwarded to the controller increases when $MC_{thr} = 0.3$ compared to $MC_{thr} = 0.2$, showing increments of 28.8%, 20.8%, and 33.6% for BruteForce, DoS/DDoS, and Botnet attacks, respectively. Additionally, the data illustrates that over 95% of sub-flows forwarded to the CP-IDS were accurately

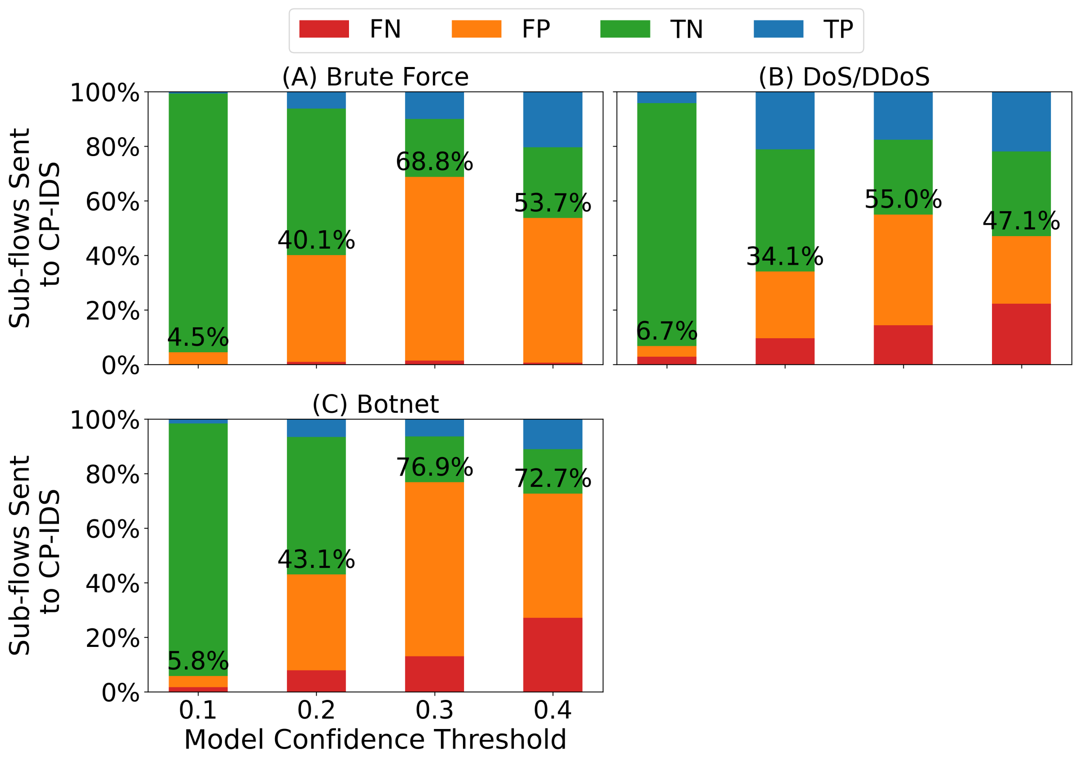

Figure 23: Percentages of various classified types of sub-flows among all sub-flows forwarded to the CP-IDS vary based on different model confidence thresholds. Specifically, the percentage of the forwarded FP and FN that are correctly forwarded to the CP-IDS is depicted in the figures. *The similar figure is in [61].*

Table 17: Comparison of important metrics to select the best MC_{thr}. Each cell represents the corresponding value for the BruteForce, DoS/DDoS, and Botnet attacks, respectively. *The table is extracted from [61].*

MC_{thr}	Forwarded Sub-flows (%)	Macro-Average F1 Score (%)	Misclassified Forwarded Flows (%)
0.1	93.5, 90.1, 70.1	96.8, 83, 95.5	4.5, 6.7, 5.8
0.2	10.1, 13.4, 8	96.7, 83.7, 96.6	40.1, 33.6, 43.2
0.3	***5.7, 7.4, 4.3***	***96.7, 87.0, 96.7***	***68.9, 54.4, 76.8***
0.4	2.8, 4.9, 2.1	88.1, 82.0, 94.7	53.8, 47.1, 72.5

detected (TP and TN). Hence, employing $MC_{thr} = 0.1$ results in the lowest efficiency regarding forwarding incorrectly classified sub-flows to the CP-IDS. In summary, Table 17 presents all the relevant metrics and their corresponding values. Each cell in the table contains three values representing the metrics for BruteForce, DoS/DDoS, and Botnet attacks. The values for $MC_{thr} = 0.3$ are highlighted in bold to facilitate comparison.

The following points lead us to choose $MC_{thr} = 0.3$ as an optimal threshold for CML-IDS.

- A MC_{thr} of 0.3 yields an average high macro-average F1 Score of 93.4% across three datasets, surpassing the scores achieved by MC_{thr} values of 0.2 and 0.4. This indicates that 0.3 offers the highest detection performance among all evaluated thresholds, except for 0.1, which is deemed impractical due to its reliance solely on ML models in the control plane.

- The detection latency when using $MC_{thr} = 0.3$ (0.361 s) is lower than that observed with 0.2 (0.546 s) and comparable to that of 0.4 (0.354 s). However, 0.4 results in the lowest detection performance among all thresholds evaluated.

- Employing $MC_{thr} = 0.3$ results in a minimal percentage of flows forwarded to the controller compared to all classified flows (5.7%, 7.5%, 4.3% for each attack type). Furthermore, it also leads to the highest proportion of incorrectly classified flows being routed to the controller, indicating that 0.3 achieves the highest efficiency in terms of flow delivery.

5.8.7 *Detection Performance of CP-IDS for Low-confident Flows*

Figure 24 illustrates the detection distribution carried out by the CP-IDS in the controller, utilizing a MC_{thr} of 0.3. This classification distribution showcases how the collaborative flow classification implemented by CML-IDS effectively addresses incorrect predictions made by the DP-IDS in the programmable switch.

For the Brute Force attack type, out of 86 flows initially misclassified as FN within the switch, 84 are rectified to TP by the CP-IDS, and over 50% of the FP predictions are corrected to TN. Furthermore, the majority of correct TN and TP predictions made within the switch remain unchanged post-classification in the CP-IDS. Notably, the collaborative classification mechanism demonstrates significant proficiency in rectifying FP to TN across all assessed attack types.

These findings underscore the efficacy of this prediction correction mechanism facilitated by collaborative flow classification in enhancing intrusion detection performance.

5.8.8 *Expiration and Hash Collision*

The CML-IDS system has a single register that acts as a buffer to store all sub-flow entries, each indexed by its unique ID. These IDs are generated using the CRC32 hash function. However, there exists a potential for hash collisions to arise, particularly when a freshly extracted sub-flow shares the same ID as an already existing sub-flow within the buffer. Hash collisions influence both the feature update and inference procedures. If a hash collision occurs during the feature updating process for a flow with less than 8 packets, it can lead to an inaccurate feature update. Moreover, if a hash collision occurs, it can result in potentially incorrect packet inference. As a result, a higher number of hash collisions indicates a more unstable system.

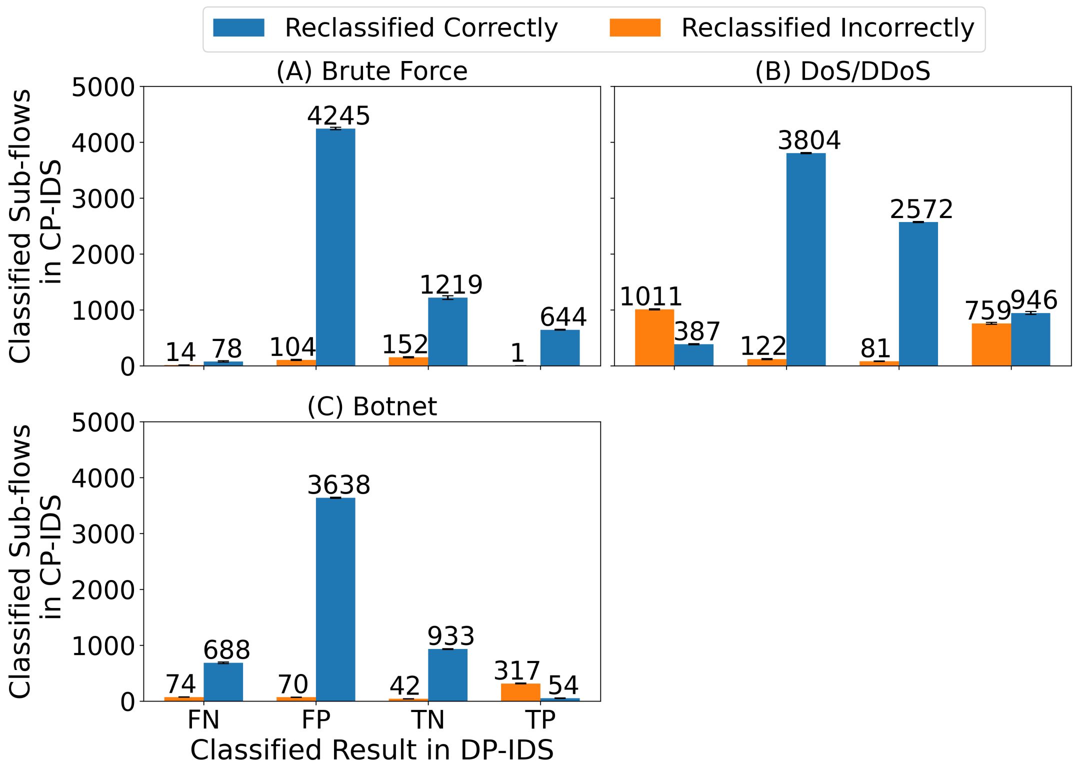

Figure 24: The re-classification distribution achieved by the CP-IDS varies for different network traffic datasets when employing a model confidence threshold of 0.3.

To ensure efficient storage of sub-flows and minimize hash collisions, a flow expiration mechanism has been implemented. This mechanism removes a sub-flow entry from the buffer after 30 minutes. The time frame of 30 minutes is aligned with the NFStream timeout for aggregating packets and creating flows, as explained in Section 5.7.2. NFStream uses sub-flow statistical features to analyze network traffic. We conducted three separate experiments to measure the frequency of hash collisions while utilizing the flow expiration mechanism.

Figure 25 demonstrates the proportion of expired flows out of all extracted flows, with an expiration timeout set at 30 minutes. Additionally, it highlights the count of flows that encountered hash collisions while stored in the flow buffer. The findings indicate that the proposed flow expiration mechanism effectively mitigated hash collisions for each attack dataset. The observed percentages of flows affected by hash collisions were 0.13%, 0.13%, and 0.30% for BruteForce, DoS/DDoS, and Botnet attacks, respectively, indicating the minimal impact of hash collisions on the overall system performance. Furthermore, 20.08%, 23.68%, and 15.83% of the flows for BruteForce, DoS/DDoS, and Botnet attacks, respectively, expired after reaching the expiration timeout, suggesting that the evaluations were conducted under stable conditions.

5.8.9 *Comparison between CML-IDS and an Existing Approach*

This section is taken almost verbatim from [61].

In this section, we provide a comparison between the CML-IDS and SwitchTree [98] which is presented in Table 18. We decided to compare with SwitchTree primarily

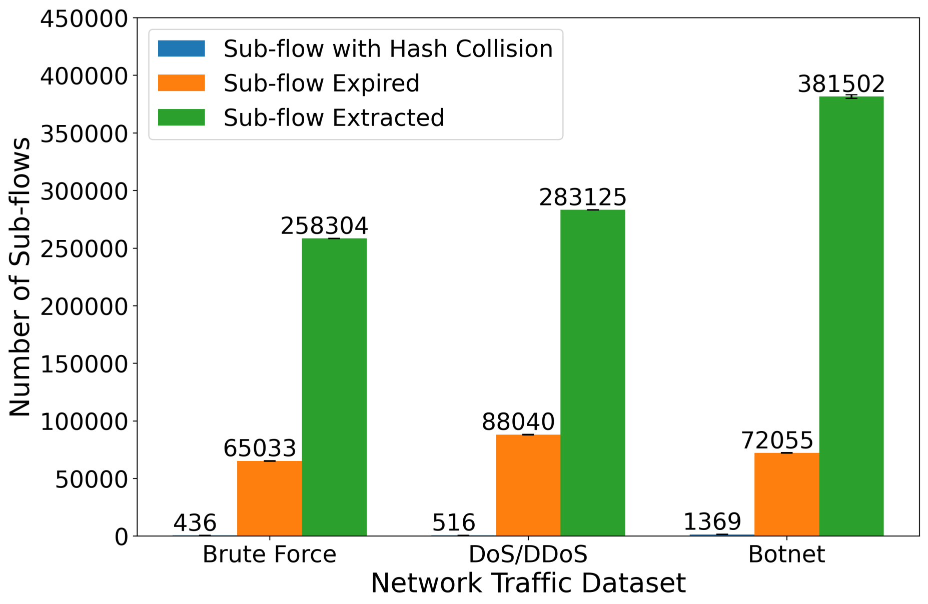

Figure 25: The number of expired flows due to the packet arrival time termination and the hash collision occurrence within the CML-IDS. These numbers are depicted in addition to the total number of extracted sub-flows.

Table 18: Comparison between CML-IDS and SwitchTree. *The table is extracted from [61].*

Metric	**CML-IDS**	**SwitchTree [98]**
Deployed ML model	CP & DP	DP
DP-IDS complexity	Max. Depth: 5	Max. Depth: 11
Number of Registers	1	20
Macro-Average F1 Score	93.4%	87.7%

because their open-source code is readily accessible, and their approach bears the closest resemblance to our DP-IDS (though not the entire CML-IDS).

For a fair comparison, it was essential to evaluate their model using the dataset employed in this study and convert the network traffic to the feature set using our proposed preprocessing pipeline. Table 18 presents a comparison of these two approaches across various metrics. The CML-IDS involves the collaboration of two ML models, DP- and CP-IDS, while SwitchTree utilizes only an RF model as a DP-IDS. Therefore, CML-IDS employs a more intricate approach compared to SwitchTree. However, the proposed DP-IDS in SwitchTree utilizes a more complex design of an RF model with a maximum depth of 11, making it challenging to embed in a programmable switch, whereas CML-IDS utilizes a lightweight RF model with a maximum depth of 5. Moreover, concerning hardware resources, the SwitchTree approach requires 20 registers, whereas CML-IDS employs an efficient register usage technique by applying the bit concatenation operation (as explained in Section 5.5.1). Additionally, the average detection performance across all attack datasets demonstrates that the CML-IDS model (with $MC_{thr} = 0.3$) outperforms SwitchTree. Furthermore, in CML-IDS, the RF model is deployed in DP-IDS with the flexibility to facilitate model updates by utilizing a combination of Machine Learning, P4 Generator, and Control Data Plane Modules. However, SwitchTree rigidly hardcoded the RF model in the software switch.

5.9 SUMMARY

In this chapter, we introduce CML-IDS (Collaborative ML-based Intrusion Detection System), a novel ML-based NIDS tailored for SDN environments. CML-IDS leverages collaboration between distinct ML models in both the data plane (DP-IDS) and control plane (CP-IDS) to achieve high detection performance, swift detection speed, and reduced network load. This collaboration is facilitated by assessing the confidence of the DP-IDS model in classifying a sub-flow and comparing it to a predefined threshold. When the model confidence is low, indicating uncertainty in classification, the sub-flow features are forwarded to the control plane for further analysis. The evaluation underscores the critical role of selecting an appropriate predefined threshold (MC_{thr}) value. In this study, a value of 0.3 is chosen for MC_{thr} based on several criteria, including detection performance, detection speed, network latency, and correct forwarding of incorrectly detected sub-flows. The results demonstrate that employing CML-IDS leads to an average reduction of 54.66% in the misclassification rate compared to the baseline, which relies solely on DP-IDS. Additionally, CML-IDS effectively enhances detection performance and reduces latency by minimizing the need to forward flows to the control plane.

6 SUMMARY, CONCLUSIONS, AND OUTLOOK

To summarize this work, an overview of the preceding chapters is presented in Section 6.1, outlining the main contributions. Drawing from the obtained results, conclusions are synthesized in Section 6.1.2. Finally, open issues and potential future work that can be done in the field of ML-based NIDSs are discussed in Section 6.2.

6.1 SUMMARY OF THE THESIS

Chapter 1 delves into the challenges faced by Machine Learning (ML)-based Network Intrusion Detection Systems (NIDS). These encompass grappling with the diversity and imbalance of network traffic datasets, which can significantly impact the generalization of feature selection and the overall detection performance of ML-based NIDSs. This influence applies not only to individual datasets sharing similar network traffic patterns but also to multiple datasets exhibiting varying traffic patterns. Additionally, the challenges associated with integrating ML-based NIDSs into programmable networks, such as in Software-Defined Networking (SDN), are discussed.

Chapter 2 provides crucial background information on the ML models utilized in this thesis, alongside an exposition of the concepts explored in each contribution chapter. Furthermore, existing research with similar objectives is investigated, identifying some research gaps. Utilizing insights from state-of-the-art analyses and considering various specific scenarios, the contributions of this thesis are presented and discussed as follows.

6.1.1 *Contributions*

Chapter 3 introduces the initial contribution, focusing on diminishing noise and feature dimensionality within network traffic datasets to reduce ML model complexity while maintaining high detection performance. Within this chapter, an Ensemble Feature Selection (EFS) method comprising a preprocessing pipeline and feature selection approach is introduced [57]. This method aims to reduce feature dimensionality across various network traffic datasets encompassing different network attack types and architectures. The results demonstrate a 65% reduction in feature dimensionality while maintaining high detection performance comparable to using all flow features for training. Through exploratory data analysis, the distinctiveness of network traffic datasets and their unique sets of relevant features are highlighted. Furthermore, the investigation into the possibility of selecting transferable features across diverse network traffic datasets through the integration of a data-driven solution (DD-EFS) is undertaken. Utilizing DD-EFS, five common features among the top 25 features across these datasets are identified, and all ML models are trained solely on these features. The findings suggest that these selected five features are more transferable across the available and previously unseen network traffic patterns.

The exploratory data analysis revealed that each network traffic dataset possesses distinct characteristics, leading to achieving high detection performance solely on network traffic patterns similar to the training dataset. Therefore, designing a model capable of accurately detecting distinct, previously unseen traffic patterns is necessary to demonstrate the generalization of its detection performance.

In response to these insights, Chapter 4 introduces a self-supervised contrastive learning approach (SSCL-NIDS) [58], which exclusively trains on benign flows. This approach enhances the capability to detect previously unseen attack flows without direct training on them, aiming to strengthen the generalization capability of the ML-based NIDS. Results demonstrate an improvement in classifying previously unseen network traffic patterns compared to both supervised and unsupervised baselines. Furthermore, the possibility of fine-tuning the SSCL-NIDS and employing it in transfer learning is explored, thereby reducing reliance on large amounts of annotated training data.

Chapter 5 investigates the deployment of ML-based NIDS within programmable network architectures like in SDN. While the control plane in SDN offers ample computational resources suitable for preprocessing and training ML models, forwarding flows to the control plane for intrusion detection tasks can lead to increased network load and detection time, which are crucial metrics to consider. To achieve line-rate detection speed and minimize network load, deploying a lightweight ML model in the programmable data plane is a viable option, given its computational resource constraints. However, the detection performance of such lightweight models may not match that of more complex ML models. To reconcile high detection performance and speed without overwhelming the control plane, a Collaborative ML-based IDS (CML-IDS) [61] is proposed. In this framework, each flow undergoes initial detection using a lightweight ML model deployed in the programmable data plane. If the detection confidence falls below a predefined threshold, the flow is forwarded to the control plane for classification using an ensemble ML model. Results demonstrate that compared to solely deploying an ML model in the data plane, the proposed CML-IDS enhances detection performance. Moreover, compared to deploying an ML model solely in the control plane, it reduces network load and detection time.

6.1.2 *Conclusions*

Throughout a comprehensive evaluation, various scenarios are examined to assess each contribution from diverse perspectives. The following summarizes the attained results.

In Section 3.6, the effectiveness of the proposed EFS approach is demonstrated by reducing feature dimensionality by at least 56% while maintaining high detection performance. To ensure the robustness of the results across different ML model structures, three different ML models are employed to evaluate the proposed EFS method:

Random Forest, Logistic Regression, and Multi-Layer Perceptron Model. Additionally, to showcase the versatility of the designed EFS pipeline, the evaluation is conducted on five distinct network traffic datasets. Across all datasets and ML models, it is observed that drastically reducing feature dimensionality can lead to a decrease in detection performance due to the potential exclusion of informative features. The results indicate that training ML models with 44% of the entire features maintain consistent detection performance. Moreover, the findings demonstrate the impact of feature dimension reduction on training time, which is a crucial metric when retraining ML models is required. Furthermore, integrating a data-driven approach into the proposed EFS method demonstrates an enhancement in the transferability of the selected features for classifying different, previously unseen network traffic patterns.

Section 4.8 presents the findings of the proposed SSCL-NIDS, which is trained exclusively on benign flows. This approach aims to reduce the annotation process for network traffic datasets and address imbalanced network traffic datasets. The results indicate that SSCL-NIDS surpasses both the supervised baseline (by over 27%) and the unsupervised baseline (by over 15%), demonstrating its capacity to learn an abstract representation of benign network traffic. This enables it to effectively identify previously unseen attack patterns. The evaluation of SSCL-NIDS involves three key scenarios: assessing detection performance on an unseen dataset extracted from the same distribution as the training dataset, evaluating performance on unseen attack flows (where the model was trained solely on benign flows from the dataset containing those attacks), and testing on an unseen attack flows dataset where the model had not encountered any flows, including benign ones. This comprehensive evaluation underscores the robustness and adaptability of SSCL-NIDS in detecting previously unseen attack patterns. Furthermore, SSCL-NIDS exhibits promising results when employed as a pretraining model and fine-tuned with only a few samples (10^{-5} of the entire dataset) from a new network traffic dataset. Remarkably, this fine-tuning process yielded a detection performance exceeding 80%, which is at least 15% higher than that of the supervised baseline model across various network traffic datasets.

Section 5.6 demonstrates a comprehensive evaluation of the proposed CML-IDS, emphasizing its detection performance, detection time, and the flows transmitted between the data plane and control plane for the classification task.

The proposed CML-IDS is compared with scenarios where ML-based NIDS is deployed solely in the control plane and solely in the programmable data plane. The findings demonstrate that CML-IDS outperformed ML-based NIDS deployed solely in the data plane, resulting in an average intrusion detection performance increase of 7.1%. Additionally, CML-IDS reduces detection time compared to ML-based NIDS deployed solely in the control plane and decreases network load by reducing packet transfers between the data plane and the control plane by approximately 78.76%. Furthermore, as the CML-IDS operates based on the confidence of the ML model deployed in the programmable data plane, the analysis examines the impact of the model confidence threshold on various metrics. This analysis aims to determine an optimal threshold for maximizing detection performance, minimizing detection time, and optimizing the percentage of forwarded flows to the control plane. Additionally, assessing the percentage of incorrectly detected flows forwarded correctly to the control plane provided valuable insights into identifying an optimal model confidence threshold.

6.2 OUTLOOK

In this work, fundamental approaches are introduced to address critical research gaps in integrating ML models into NIDSs.

The first contribution emphasizes reducing feature dimensionality to mitigate noise in network traffic datasets. Additionally, a transferable feature set among various network traffic datasets is selected through the integration of a data-driven approach. Based on the results, the proposed EFS and DD-EFS methods can be employed for online learning approaches, facilitating rapid retraining of the ML model [60]. Moreover, as EFS is utilized in the preprocessing step for ML-based NIDS, it can be integrated into more complex ML-based NIDSs, as demonstrated in the second and third contributions of this thesis.

In the second contribution, an approach called SSCL-NIDS is devised to be exclusively trained on benign flows, aiming to enhance the generalization of detection performance in ML-based NIDSs. Data augmentation is accomplished by applying a random corruption mask to the flow sample within the proposed SSCL-NIDS. To enable the model to acquire a more generic representation of benign flows and improve its detection performance against adversarial attack samples, it is feasible to employ Generative Adversarial Networks to generate negative samples.

Furthermore, the SSCL-NIDS is utilized as a pre-trained model and fine-tuned with a limited number of samples from new network traffic patterns. This strategy enables the method to effectively classify new network patterns, even with only a small number of samples. Leveraging transfer learning eliminates the necessity to train a model from scratch, as the SSCL-NIDS is already trained with a diverse range of network traffic patterns, thereby enhancing its adaptability in different scenarios.

The final contribution focuses on facilitating the integration of ML-based NIDS into SDN, considering factors such as detection performance, speed, and potential additional network load. The framework is implemented within a softwarized P4 switch (BMv2), taking into consideration potential hardware limitations, such as those of Tofino switches. As a follow-up work, investigating the deployment of the framework on real hardware will be conducted to confirm its practical viability. Moreover, the flexible design of CML-IDS allows for the addition of new modules aimed at retraining deployed ML models and integrating online learning approaches. This has the potential to enhance the detection performance of CML-IDS when encountering various previously unseen traffic patterns.

OPEN-SOURCE

Collaboration between researchers and engineers drives scientific advancements and technological innovations. To foster future achievements within the research community and enable the replication of our findings, we have openly shared multiple implementations of this work as open-source projects.

ACKNOWLEDGMENTS

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) through CELTIC-NEXT Flagship Project AI-NET-PROTECT and the Software Campus Grant 01IS17050 (ML-based NIDS).

BIBLIOGRAPHY

[1] Oluwadamilare Harazeem Abdulganiyu, Taha Ait Tchakoucht, and Yakub Kayode Saheed. "A systematic literature review for network intrusion detection system (IDS)." In: *International Journal of Information Security* 22.5 (2023), pp. 1–38. DOI: 10.1007/s10207-023-00682-2.

[2] Manal Abdullah, Arwa Alshannaq, Asmaa Balamash, and Soad Almabdy. "Enhanced intrusion detection system using feature selection method and ensemble learning algorithms." In: *International Journal of Computer Science and Information Security (IJCSIS)* 16.2 (2018), pp. 48–55.

[3] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. "Network intrusion detection system: A systematic study of machine learning and deep learning approaches." In: *Transactions on Emerging Telecommunications Technologies* 32.1 (2021), e4150. DOI: 10.1002/ett.4150.

[4] Saleh Albelwi. "Survey on Self-Supervised Learning: Auxiliary Pretext Tasks and Contrastive Learning Methods in Imaging." In: *Journal of Entropy* 24.4 (2022). DOI: 10.3390/e24040551.

[5] Iñaki Aldasoro, Leonardo Gambacorta, Paolo Giudici, and Thomas Leach. "The drivers of cyber risk." In: *Journal of Financial Stability* 60 (2022), p. 100989. DOI: 10.1016/j.jfs.2022.100989.

[6] Osama Alkadi, Nour Moustafa, and Benjamin Turnbull. "A Review of Intrusion Detection and Blockchain Applications in the Cloud: Approaches, Challenges and Solutions." In: *Journal of IEEE Access* 8 (2020), pp. 104893–104917. DOI: 10.1109/ACCESS.2020.2999715.

[7] Sarah Alkadi, Saad Al-Ahmadi, and Mohamed Maher Ben Ismail. "Toward Improved Machine Learning-Based Intrusion Detection for Internet of Things Traffic." In: *Journal of Computers* 12.8 (2023). DOI: 10.3390/computers12080148.

[8] Mohammad Almseidin, Jamil Al-Sawwa, Mouhammd Alkasassbeh, and Mohammed Alweshah. "On detecting distributed denial of service attacks using fuzzy inference system." In: *Journal of Cluster Computing* 26.2 (2023), pp. 1337–1351. DOI: 10.1007/s10586-022-03657-5.

[9] Zahangir Alom and Tarek Taha. "Network intrusion detection for cyber security using unsupervised deep learning approaches." In: *IEEE National Aerospace and Electronics Conference (NAECON)*. 2017, pp. 63–69. DOI: 10.1109/NAECON.2017.8268746.

[10] Fatemeh Amiri, MohammadMahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. “Mutual information-based feature selection for intrusion detection systems.” In: *Journal of Network and Computer Applications* 34.4 (2011). Advanced Topics in Cloud Computing, pp. 1184–1199. DOI: 10.1016/j.jnca.2011.01.002.

[11] Zied Aouini and Adrian Pekar. “NFStream: A flexible network data analysis framework.” In: *Journal of Computer Networks* 204 (2022), p. 108719. DOI: 10.1016/j.comnet.2021.108719.

[12] Francisco Aparicio-Navarro, Konstantinos Kyriakopoulos, Ibrahim Ghafir, Sangarapillai Lambotharan, and Jonathon Chambers. “Multi-Stage Attack Detection Using Contextual Information.” In: *IEEE Military Communications Conference (MILCOM)*. 2018, pp. 1–9. DOI: 10.1109/MILCOM.2018.8599708.

[13] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. “SNAP: Stateful Network-Wide Abstractions for Packet Processing.” In: *Proceedings of the ACM SIGCOMM Conference*. Association for Computing Machinery, 2016, pp. 29–43. DOI: 10.1145/2934872.2934892.

[14] *AWS hit by largest reported DDoS attack of 2.3 Tbps*. June 2020. URL: https://www.a10networks.com/blog/aws-hit-by-largest-reported-ddos-attack-of-2-3-tbps/ (Last accessed on April 10, 2024).

[15] Zahedi Azam, Md. Motaharul Islam, and Mohammad Nurul Huda. “Comparative Analysis of Intrusion Detection Systems and Machine Learning-Based Model Analysis Through Decision Tree.” In: *Journal of IEEE Access* 11 (2023), pp. 80348–80391. DOI: 10.1109/ACCESS.2023.3296444.

[16] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali Ghorbani. “Towards effective feature selection in machine learning-based botnet detection approaches.” In: *IEEE Conference on Communications and Network Security*. 2014, pp. 247–255. DOI: 10.1109/CNS.2014.6997492.

[17] Andrea Bommert, Xudong Sun, Bernd Bischl, Jörg Rahnenführer, and Michel Lang. “Benchmark for filter methods for feature selection in high-dimensional classification data.” In: *Journal of Computational Statistics & Data Analysis* 143 (2020), p. 106839. DOI: 10.1016/j.csda.2019.106839.

[18] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. “P4: programming protocol-independent packet processors.” In: *SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95. DOI: 10.1145/2656877.2656890.

[19] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. “Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN.” In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. Association for Computing Machinery, 2013, pp. 99–110. DOI: 10.1145/2486001.2486011.

[20] Fatima Bouchama and Mostafa Kamal. "Enhancing Cyber Threat Detection through Machine Learning-Based Behavioral Modeling of Network Traffic Patterns." In: *International Journal of Business Intelligence and Big Data Analytics* 4.9 (2021), pp. 1–9. URL: `https://research.tensorgate.org/index.php/IJBIBDA/article/view/76` (Last accessed on April 10, 2024).

[21] Leo Breiman. "Bagging predictors." In: *Journal of Machine learning* 24 (1996), pp. 123–140. DOI: `10.1007/BF00058655`.

[22] Leo Breiman. "Random forests." In: *Journal of Machine learning* 45.1 (2001), pp. 5–32. DOI: `10.1023/A:1010933404324`.

[23] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. *pForest: In-Network Inference with Random Forests*. 2022. DOI: `10.48550/arXiv.1909.05680`.

[24] Ranyelson Neres Carvalho, Lucas Rodrigues Costa, Jacir Luiz Bordim, and Eduardo Adilio Pelinson Alchieri. "Detecting DDoS Attacks on SDN Data Plane with Machine Learning." In: *Ninth International Symposium on Computing and Networking Workshops (CANDARW)*. 2021, pp. 138–144. DOI: `10.1109/CANDARW53999.2021.00030`.

[25] Carlos Catania and Carlos García Garino. "Automatic network intrusion detection: Current techniques and open issues." In: *Journal of Computers & Electrical Engineering* 38.5 (2012). Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing, pp. 1062–1072. DOI: `10.1016/j.compeleceng.2012.05.013`.

[26] Evan Caville, Wai Weng Lo, Siamak Layeghy, and Marius Portmann. "Anomal-E: A self-supervised network intrusion detection system based on graph neural networks." In: *Knowledge-Based Systems Journal* 258 (2022), p. 110030. DOI: `10.1016/j.knosys.2022.110030`.

[27] David Charte, Francisco Charte, María J. del Jesus, and Francisco Herrera. "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges." In: *Journal of Neurocomputing* 404 (2020), pp. 93–107. DOI: `10.1016/j.neucom.2020.04.057`.

[28] Nitesh Chawla, Kevin Bowyer, Lawrence Hall, and Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357. DOI: `10.1613/jair.953`.

[29] Kuan-yin Chen, Anudeep Reddy Junuthula, Ishant Kumar Siddhrau, Yang Xu, and H. Jonathan Chao. "SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane." In: *2016 IEEE Conference on Communications and Network Security (CNS)*. 2016, pp. 28–36. DOI: `10.1109/CNS.2016.7860467`.

[30] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[31] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. DOI: 10.48550/arXiv.2002.05709. arXiv: 2002.05709 [cs.LG].

[32] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. "A Simple Framework for Contrastive Learning of Visual Representations." In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. PMLR, 2020, pp. 1597–1607. URL: https://proceedings.mlr.press/v119/chen20j.html (Last accessed on April 10, 2024).

[33] Hyunseung Choi, Mintae Kim, Gyubok Lee, and Wooju Kim. "Unsupervised learning approach for network intrusion detection system using autoencoders." In: *The Journal of Supercomputing* 75 (2019), pp. 5597–5621. DOI: 10.1007/s11227-019-02805-w.

[34] *Cisco Annual Internet Report (2018-2023)*. Mar. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (Last accessed on April 10, 2024).

[35] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero Vega. "Security in SDN: A comprehensive survey." In: *Journal of Network and Computer Applications* 159 (2020), p. 102595. DOI: 10.1016/j.jnca.2020.102595.

[36] Weverton Luis da Costa Cordeiro, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. "Data plane programmability beyond openflow: Opportunities and challenges for network and service operations and management." In: *Journal of Network and Systems Management* 25 (2017), pp. 784–818. DOI: 10.1007/s10922-017-9423-2.

[37] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. "Supervised Learning." In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Springer Berlin Heidelberg, 2008, pp. 21–49. DOI: 10.1007/978-3-540-75171-7_2.

[38] Laurens D'hooge, Miel Verkerken, Tim Wauters, Filip De Turck, and Bruno Volckaert. "Investigating Generalized Performance of Data-Constrained Supervised Machine Learning Models on Novel, Related Samples in Intrusion Detection." In: *Journal of Sensors* 23.4 (2023). DOI: 10.3390/s23041846.

[39] Laurens D'hooge, Tim Wauters, Bruno Volckaert, and Filip De Turck. "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection." In: *Journal of Information Security and Applications* 54 (2020), p. 102564. DOI: 10.1016/j.jisa.2020.102564.

[40] Shaveta Dargan, Munish Kumar, Maruthi Rohit Ayyagari, and Gulshan Kumar. "A survey of deep learning and its applications: a new paradigm to machine learning." In: *Archives of Computational Methods in Engineering Journal* 27 (2020), pp. 1071–1092. DOI: 10.1007/s11831-019-09344-w.

[41] Raktim Deb and Sudipta Roy. "A comprehensive survey of vulnerability and information security in SDN." In: *Journal of Computer Networks* 206 (2022), p. 108802. DOI: 10.1016/j.comnet.2022.108802.

[42] Ayesha Siddiqua Dina and Dakshnamoorthy Manivannan. "Intrusion detection based on Machine Learning techniques in computer networks." In: *Journal of Internet of Things* 16 (2021), p. 100462. DOI: 10.1016/j.iot.2021.100462.

[43] Abhimanyu Dubey, Vignesh Ramanathan, Alex Pentland, and Dhruv Mahajan. "Adaptive Methods for Real-World Domain Generalization." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 14340–14349. DOI: 10.1109/CVPR46437.2021.01411.

[44] Heba F. Eid, Aboul Ella Hassanien, Tai-hoon Kim, and Soumya Banerjee. "Linear Correlation-Based Feature Selection for Network Intrusion Detection Model." In: *Advances in Security of Information and Communication Networks*. Springer Berlin Heidelberg, 2013, pp. 240–248. DOI: 10.1007/978-3-642-40597-6_21.

[45] Lubna Fayez Eliyan and Roberto Di Pietro. "DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges." In: *Journal of Future Generation Computer Systems* 122 (2021), pp. 149–171. DOI: 10.1016/j.future.2021.03.011.

[46] Fahimeh Farahnakian and Jukka Heikkonen. "A deep auto-encoder based approach for intrusion detection system." In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. 2018, pp. 178–183. DOI: 10.23919/ICACT.2018.8323688.

[47] Hongliang Fei, Brian Quanz, and Jun Huan. "Regularization and feature selection for networked features." In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. Association for Computing Machinery, 2010, pp. 1893–1896. DOI: 10.1145/1871437.1871756.

[48] Qusyairi Ridho Saeful Fitni and Kalamullah Ramli. "Implementation of Ensemble Learning and Feature Selection for Performance Improvements in Anomaly-Based Intrusion Detection Systems." In: *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. 2020, pp. 118–124. DOI: 10.1109/IAICT50021.2020.9172014.

[49] Thangasamy Anitha G. Logeswari Shilpi Bose. "An Intrusion Detection System for SDN Using Machine Learning." In: *Journal of Intelligent Automation & Soft Computing* 35.1 (2023), pp. 867–880. DOI: 10.32604/iasc.2023.026769.

[50] Mudasir Ahmad Ganaie, Minghui Hu, Ashwani Kumar Malik, Mohammad Tanveer, and Ponnuthurai Nagaratnam Suganthan. "Ensemble deep learning: A review." In: *Journal of Engineering Applications of Artificial Intelligence* 115 (2022), p. 105151. DOI: 10.1016/j.engappai.2022.105151.

[51] Aparna Ganesan and Kamil Sarac. "Mitigating Evasion Attacks on Machine Learning based NIDS Systems in SDN." In: *IEEE 7th International Conference on Network Softwarization (NetSoft)*. 2021, pp. 268–272. DOI: 10.1109/NetSoft51509.2021.9492526.

[52] Sebastian García, Martin Grill, Jan Stiborek, and Alejandro Zunino. "An empirical comparison of botnet detection methods." In: *Journal of Computers & Security* 45 (2014), pp. 100–123. DOI: 10.1016/j.cose.2014.05.011.

[53] Pedro García-Teodoro, Jesús Díaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges." In: *Journal of Computers & Security* 28.1 (2009), pp. 18–28. DOI: 10.1016/j.cose.2008.08.003.

[54] Christoph Gärtner, Amr Rizk, Boris Koldehofe, Rhaban Hark, René Guillaume, Ralf Kundel, and Ralf Steinmetz. "POSTER: Leveraging PIFO Queues for Scheduling in Time-Sensitive Networks." In: *2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE, 2021, pp. 1–2. DOI: 10.1109/LANMAN52105.2021.9478796.

[55] Sravanthi Godala and Rama Prasad Venkata Vaddella. "A study on intrusion detection system in wireless sensor networks." In: *International Journal of Communication Networks and Information Security* 12.1 (2020), pp. 127–141. DOI: 10.17762/ijcnis.v12i1.4429.

[56] Pegah Golchin, Leonard Anderweit, Julian Zobel, Ralf Kundel, and Ralf Steinmetz. "In-Network SYN Flooding DDoS Attack Detection Utilizing P4 Switches." In: *Proceedings of the 3rd KuVS Fachgespräch "Network Softwarization"*. 2022, pp. 1–2. DOI: 10.15496/publikation-67441.

[57] Pegah Golchin, Ralf Kundel, Tim Steuer, Rhaban Hark, and Ralf Steinmetz. "Improving DDoS Attack Detection Leveraging a Multi-aspect Ensemble Feature Selection." In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022, pp. 1–5. DOI: 10.1109/NOMS54207.2022.9789763.

[58] Pegah Golchin, Nima Rafiee, Mehrdad Hajizadeh, Ahmad Khalil, Ralf Kundel, and Ralf Steinmetz. "SSCL-IDS: Enhancing Generalization of Intrusion Detection with Self-Supervised Contrastive Learning." In: *2024 IFIP Networking Conference (IFIP Networking)*. IEEE. 2024, pp. 1–9. Forthcoming.

[59] Pegah Golchin, Nima Rafiee, and Ralf Kundel. "A Data-Driven Solution for Improving Transferability of Traffic Flow Feature Selection." In: *2024 IFIP Networking Conference (IFIP Networking)*. IEEE. 2024, pp. 1–3. Forthcoming.

[60] Pegah Golchin, Jannis Weil, Ralf Kundel, and Ralf Steinmetz. “Dynamic network intrusion detection system in Software-Defined Networking.” In: *2nd Workshop on Machine Learning & Networking (MaLeNe), co-located with the 5th International Conference on Networked Systems (NetSys 2023)*. 2023, pp. 1–2.

[61] Pegah Golchin, Chengbo Zhou, Pratyush Agnihotri, Pratyush Agnihotri, Mehrdad Hajizadeh, Ralf Kundel, and Ralf Steinmetz. “CML-IDS: Enhancing Intrusion Detection in SDN Through Collaborative Machine Learning.” In: *19th International Conference on Network and Service Management (CNSM)*. 2023, pp. 1–9. DOI: `10.23919/CNSM59352.2023.10327863`.

[62] Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. “CorrCorr: A feature selection method for multivariate correlation network anomaly detection techniques.” In: *Journal of Computers & Security* 83 (2019), pp. 234–245. DOI: `10.1016/j.cose.2019.02.008`.

[63] Palash Goyal, Sumit Pandey, and Karan Jain. *Deep Learning for Natural Language Processing*. 1st. Apress Berkeley, CA, 2018. DOI: `10.1007/978-1-4842-3685-7`.

[64] The P4.org API Working Group. *P4Runtime Specification*. `https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html`. (Last accessed on April 10, 2024).

[65] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection.” In: *Journal of Machine Learning Research* 3 (2003), pp. 1157–1182. DOI: `10.5555/944919.944968`.

[66] Nabil Hachem, Yosra Ben Mustapha, Gustavo Gonzalez Granadillo, and Herve Debar. “Botnets: Lifecycle and Taxonomy.” In: *Conference on Network and Information Systems Security*. 2011, pp. 1–8. DOI: `10.1109/SAR-SSI.2011.5931395`.

[67] Mehrdad Hajizadeh, Sudip Barua, and Pegah Golchin. “FSA-IDS: A Flow-based Self-Active Intrusion Detection System.” In: *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2023, pp. 1–9. DOI: `10.1109/NOMS56928.2023.10154343`.

[68] Suzan Hajj, Rayane El Sibai, Jacques Bou Abdo, Jacques Demerjian, Abdallah Makhoul, and Christophe Guyeux. “Anomaly-based intrusion detection systems: The requirements, methods, measurements, and datasets.” In: *Transactions on Emerging Telecommunications Technologies* 32.4 (2021), e4240. DOI: `10.1002/ett.4240`.

[69] Simon Hansman and Ray Hunt. “A taxonomy of network and computer attacks.” In: *Journal of Computers & Security* 24.1 (2005), pp. 31–43. DOI: `10.1016/j.cose.2004.06.011`.

[70] Simon Hansman and Ray Hunt. “A taxonomy of network and computer attacks.” In: *Journal of Computers & Security* 24.1 (2005), pp. 31–43. DOI: `10.1016/j.cose.2004.06.011`.

[71] Rhaban Hark, Mohamed Ghanmi, Ralf Kundel, Patrick Lieser, and Ralf Steinmetz. "Monitoring Flows with Per-Application Granularity using Programmable Data Planes." In: *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 2021, pp. 1–6. DOI: 10.1109/LANMAN52105.2021.9478798.

[72] Rhaban Simon Hark. "Monitoring Federated Softwarized Networks: Approaches for Efficient and Collaborative Data Collection in Large-Scale Software-Defined Networks." PhD thesis. Darmstadt: Technische Universität Darmstadt, 2019. URL: http://tuprints.ulb.tu-darmstadt.de/9073/.

[73] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. "A survey on data plane programming with P4: Fundamentals, advances, and applied research." In: *Journal of Network and Computer Applications* 212 (2023), p. 103561. DOI: 10.1016/j.jnca.2022.103561.

[74] Simon Haykin. *Neural Networks and Learning Machines*. 3rd ed. Pearson Education India. URL: https://books.google.de/books?id=ivK0DwAAQBAJ (Last accessed on April 10, 2024).

[75] Imran Hidayat, Muhammad Zulfiqar Ali, and Arshad Arshad. "Machine Learning-Based Intrusion Detection System: An Experimental Comparison." In: *Journal of Computational and Cognitive Engineering* 2.2 (2022), pp. 88–97. DOI: 10.47852/bonviewJCCE2202270.

[76] Nazrul Hoque, Monowar Bhuyan, Ram Charan Baishya, Dhruba Kumar Bhattacharyya, and Jugal Kumar Kalita. "Network attacks: Taxonomy, tools and systems." In: *Journal of Network and Computer Applications* 40 (2014), pp. 307–324. DOI: 10.1016/j.jnca.2013.08.001.

[77] Asmaul Hosna, Ethel Merry, Jigmey Gyalmo, Zulfikar Alom, Zeyar Aung, and Mohammad Abdul Azim. "Transfer learning: a friendly introduction." In: *Journal of Big Data* 9.1 (2022), p. 102. DOI: 10.1186/s40537-022-00652-w.

[78] Fei Hu, Qi Hao, and Ke Bao. "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation." In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2181–2206. DOI: 10.1109/COMST.2014.2326417.

[79] Shamsul Huda, Kevin Liu, Mohamed Abdelrazek, Amani Ibrahim, Sultan Alyahya, Hmood Al-Dossari, and Shafiq Ahmad. "An Ensemble Oversampling Model for Class Imbalance Problem in Software Defect Prediction." In: *Journal of IEEE Access* 6 (2018), pp. 24184–24195. DOI: 10.1109/ACCESS.2018.2817572.

[80] MohammadNoor Injadat, Abdallah Moubayed, Ali Bou Nassif, and Abdallah Shami. "Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection." In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1803–1816. DOI: 10.1109/TNSM.2020.3014929.

[81] *IOT traffic is tracking over; 5G, WiFi 6 are ascending*. Jan. 2020. URL: https://www.networkworld.com/article/968399/cisco-iot-traffic-is-taking-over-5g-wifi-6-are-ascending.html (Last accessed on April 10, 2024).

[82] Hossein Hadian Jazi, Hugo Gonzalez, Natalia Stakhanova, and Ali Ghorbani. "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling." In: *Journal of Computer Networks* 121 (2017), pp. 25–36. DOI: 10.1016/j.comnet.2017.03.018.

[83] María Jiménez, David Fernández, Jorge Eduardo Rivadeneira, Luis Bellido, and Andrés Cárdenas. "A Survey of the Main Security Issues and Solutions for the SDN Architecture." In: *Journal of IEEE Access* 9 (2021), pp. 122016–122038. DOI: 10.1109/ACCESS.2021.3109564.

[84] Myung-Jin Jun. "A comparison of a gradient boosting decision tree, random forests, and artificial neural networks to model urban land use changes: the case of the Seoul metropolitan area." In: *International Journal of Geographical Information Science* 35.11 (2021), pp. 2149–2167. DOI: 10.1080/13658816.2021.1887490.

[85] Rajesh Kalakoti, Sven Nõmm, and Hayretdin Bahsi. "In-Depth Feature Selection for the Statistical Machine Learning-Based Botnet Detection in IoT Networks." In: *Journal of IEEE Access* 10 (2022), pp. 94518–94535. DOI: 10.1109/ACCESS.2022.3204001.

[86] Murat Karakus and Arjan Durresi. "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)." In: *Journal of Computer Networks* 112 (2017), pp. 279–293. DOI: 10.1016/j.comnet.2016.11.017.

[87] Sydney Mambwe Kasongo and Yanxia Sun. "Performance analysis of intrusion detection systems using a feature selection method on the UNSW-NB15 dataset." In: *Journal of Big Data* 7 (2020), pp. 1–20. DOI: 10.1186/s40537-020-00379-6.

[88] Elie Kfoury, Jorge Crichigno, and Elias Bou-Harb. "An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends." In: *Journal of IEEE Access* 9 (2021), pp. 87094–87155. DOI: 10.1109/ACCESS.2021.3086704.

[89] Geeta Kocher and Gulshan Kumar. "Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges." In: *Journal of Soft Computing* 25.15 (2021), pp. 9731–9763. DOI: 10.1007/s00500-021-05893-0.

[90] Deepak Kshirsagar and Sandeep Kumar. "A feature reduction based reflected and exploited DDoS attacks detection system." In: *Journal of Ambient Intelligence and Humanized Computing* 13.1 (2022), pp. 393–405. DOI: 10.1007/s12652-021-02907-5.

[91] Vinod Kumar, Vinay Choudhary, Vivek Sahrawat, and Vinay Kumar. "Detecting Intrusions and Attacks in the Network Traffic using Anomaly based Techniques." In: *5th International Conference on Communication and Electronics Systems (ICCES)*. 2020, pp. 554–560. DOI: 10.1109/ICCES48766.2020.9137968.

[92] Ralf Kundel. "Accelerating Network Functions Using Reconfigurable Hardware: Design and Validation of High Throughput and Low Latency Network Functions at the Access Edge." PhD thesis. Technische Universität Darmstadt, 2024. URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/22023.

[93] Ralf Kundel, Christoph Gärtner, Manisha Luthra, Sukanya Bhowmik, and Boris Koldehofe. "Flexible Content-based Publish/Subscribe over Programmable Data Planes." In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–5. DOI: 10.1109/NOMS47738.2020.9110381.

[94] Siamak Layeghy, Mahsa Baktashmotlagh, and Marius Portmann. "DI-NIDS: Domain invariant network intrusion detection system." In: *Journal of Knowledge-Based Systems* 273 (2023), p. 110626. DOI: 10.1016/j.knosys.2023.110626.

[95] Siamak Layeghy and Marius Portmann. "Explainable Cross-domain Evaluation of ML-based Network Intrusion Detection Systems." In: *Journal of Computers and Electrical Engineering* 108 (2023), p. 108692. DOI: 10.1016/j.compeleceng.2023.108692.

[96] Siamak Layeghy and Marius Portmann. "Explainable Cross-domain Evaluation of ML-based Network Intrusion Detection Systems." In: *Journal of Computers and Electrical Engineering* 108 (2023), p. 108692. DOI: 10.1016/j.compeleceng.2023.108692.

[97] Long Tan Le and Tran Ngoc Thinh. "On the Improvement of Machine Learning Based Intrusion Detection System for SDN Networks." In: *8th NAFOSTED Conference on Information and Computer Science (NICS)*. 2021, pp. 464–469. DOI: 10.1109/NICS54270.2021.9701522.

[98] Jong-Hyouk Lee and Kamal Singh. "Switchtree: in-network computing and traffic analyses with random forests." In: *Neural Computing and Applications* (2020), pp. 1–12. DOI: 10.1007/s00521-020-05440-2.

[99] Moemedi Lefoane, Ibrahim Ghafir, Sohag Kabir, and Irfan-Ullah Awan. "Multi-stage Attack Detection: Emerging Challenges for Wireless Networks." In: *2022 International Conference on Smart Applications, Communications and Networking (SmartNets)*. 2022, pp. 01–05. DOI: 10.1109/SmartNets55823.2022.9994027.

[100] Wenjuan Li, Weizhi Meng, and Lam For Kwok. "A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures." In: *Journal of Network and Computer Applications* 68 (2016), pp. 126–139. DOI: 10.1016/j.jnca.2016.04.011.

[101] Shan Lin, Hong Zheng, Bei Han, Yanyan Li, Chao Han, and Wei Li. "Comparative performance of eight ensemble learning approaches for the development of models of slope stability prediction." In: *Journal of Acta Geotechnica* 17.4 (2022), pp. 1477–1502. DOI: 10.1007/s11440-021-01440-1.

[102] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2018. DOI: 10.48550/arXiv.1708.02002. arXiv: 1708.02002 [cs.CV].

[103] Haoyue Liu, MengChu Zhou, and Qing Liu. "An embedded feature selection method for imbalanced data classification." In: *IEEE/CAA Journal of Automatica Sinica* 6.3 (2019), pp. 703–715. DOI: 10.1109/JAS.2019.1911447.

[104] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. "p4v: practical verification for programmable data planes." In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 2018, pp. 490–503. DOI: 10.1145/3230543.3230582.

[105] Lan Liu, Pengcheng Wang, Jianliang Ruan, and Jun Lin. "Conflow: contrast network flow improving class-imbalanced learning in network intrusion detection." In: *Research Square Preprint* (2022). DOI: 10.21203/rs.3.rs-1572776/v1.

[106] Qigang Liu, Deming Wang, Yuhang Jia, Suyuan Luo, and Chongren Wang. "A multi-task based deep learning approach for intrusion detection." In: *Journal of Knowledge-Based Systems* 238 (2022), p. 107852. DOI: 10.1016/j.knosys.2021.107852.

[107] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad Eid Alsaadi. "A survey of deep neural network architectures and their applications." In: *Journal of Neurocomputing* 234 (2017), pp. 11–26. DOI: 10.1016/j.neucom.2016.12.038.

[108] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. "Self-Supervised Learning: Generative or Contrastive." In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2023), pp. 857–876. DOI: 10.1109/TKDE.2021.3090866.

[109] Ziyu Liu, Azadeh Alavi, Minyi Li, and Xiang Zhang. "Self-Supervised Contrastive Learning for Medical Time Series: A Systematic Review." In: *Journal of Sensors* 23.9 (2023). DOI: 10.3390/s23094221.

[110] Manuel Lopez-Martin, Antonio Sanchez-Esguevillas, Juan Ignacio Arribas, and Belen Carro. "Supervised contrastive learning over prototype-label embeddings for network intrusion detection." In: *Information Fusion Journal* 79 (2022), pp. 200–228. DOI: 10.1016/j.inffus.2021.09.014.

[111] Gilles Louppe. *Understanding Random Forests: From Theory to Practice*. 2015. DOI: 10.48550/arXiv.1407.7502. arXiv: 1407.7502 [stat.ML].

[112] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions." In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. DOI: doi/10.5555/3295222.3295230.

[113] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[114] Yassine Maleh, Youssef Qasmaoui, Khalid El Gholami, Yassine Sadqi, and Soufyane Mounir. "A comprehensive survey on SDN security: threats, mitigations, and future directions." In: *Journal of Reliable Intelligent Environments* 9.2 (2023), pp. 201–239. DOI: 10.1007/s40860-022-00171-8.

[115] Ziadoon Kamil Maseer, Robiah Yusof, Nazrulazhar Bahaman, Salama Mostafa, and Cik Feresa Mohd Foozy. "Benchmarking of Machine Learning for Anomaly Based Intrusion Detection Systems in the CICIDS2017 Dataset." In: *Journal of IEEE Access* 9 (2021), pp. 22351–22370. DOI: 10.1109/ACCESS.2021.3056614.

[116] Fares Meghdouri, Tanja Zseby, and Félix Iglesias. "Analysis of Lightweight Feature Vectors for Attack Detection in Network Traffic." In: *Journal of Applied Sciences* 8.11 (2018). DOI: 10.3390/app8112196.

[117] Weizhi Meng, Wenjuan Li, and Lam-For Kwok. "EFM: Enhancing the performance of signature-based network intrusion detection systems using enhanced filter mechanism." In: *Journal of Computers & Security* 43 (2014), pp. 189–204. DOI: 10.1016/j.cose.2014.02.006.

[118] Michael Menth, Habib Mostafaei, Daniel Merling, and Marco Häberle. "Implementation and Evaluation of Activity-Based Congestion Management Using P4 (P4-ABC)." In: *Journal of Future Internet* 11.7 (2019). DOI: 10.3390/fi11070159.

[119] Bruno Henrique Meyer, Aurora Trinidad Ramirez Pozo, Michele Nogueira, and Wagner Nunan Zola. "Federated Self-Supervised Learning for Intrusion Detection." In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. 2023, pp. 822–828. DOI: 10.1109/SSCI52147.2023.10371956.

[120] Ishan Misra and Laurens van der Maaten. "Self-Supervised Learning of Pretext-Invariant Representations." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020. DOI: 10.48550/arXiv.1912.01991.

[121] Abdelrahman Mohamed, Hung-yi Lee, Lasse Borgholt, Jakob D. Havtorn, Joakim Edin, Christian Igel, Katrin Kirchhoff, Shang-Wen Li, Karen Livescu, Lars Maaløe, Tara N. Sainath, and Shinji Watanabe. "Self-Supervised Speech Representation Learning: A Review." In: *IEEE Journal of Selected Topics in Signal Processing* 16.6 (2022), pp. 1179–1210. DOI: 10.1109/JSTSP.2022.3207050.

[122] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." In: *Military Communications and Information Systems Conference (MilCIS)*. 2015, pp. 1–6. DOI: 10.1109/MilCIS.2015.7348942.

[123] Francesco Musumeci, Ali Can Fidanci, Francesco Paolucci, Filippo Cugini, and Massimo Tornatore. "Machine-Learning-enabled DDoS attacks detection in P4 programmable networks." In: *Journal of Network and Systems Management* 30.1 (2022), pp. 1–27. DOI: 10.1007/s10922-021-09633-5.

[124] Muhammad Nadeem, Ali Arshad, Saman Riaz, Shahab Band, and Amir Mosavi. "Intercept the Cloud Network From Brute Force and DDoS Attacks via Intrusion Detection and Prevention System." In: *Journal of IEEE Access* 9 (2021), pp. 152300–152309. DOI: 10.1109/ACCESS.2021.3126535.

[125] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. *A Comprehensive Overview of Large Language Models*. 2024. DOI: 10.48550/arXiv.2307.06435. arXiv: 2307.06435 [cs.CL].

[126] Hoang Nguyen and Rasha Kashef. "TS-IDS: Traffic-aware self-supervised learning for IoT Network Intrusion Detection." In: *Journal of Knowledge-Based Systems* 279 (2023), p. 110966. DOI: 10.1016/j.knosys.2023.110966.

[127] Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks." In: *IEEE Communications Surveys & Tutorials* 16.3 (2014), pp. 1617–1634. DOI: 10.1109/SURV.2014.012214.00180.

[128] Kriti Ohri and Mukesh Kumar. "Review on self-supervised image recognition using deep neural networks." In: *Journal of Knowledge-Based Systems* 224 (2021), p. 107090. DOI: 10.1016/j.knosys.2021.107090.

[129] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. DOI: 10.48550/arXiv.1807.03748. arXiv: 1807.03748 [cs.LG].

[130] Opeyemi Osanaiye, Haibin Cai, Kim-Kwang Raymond Choo, Ali Dehghantanha, Zheng Xu, and Mqhele Dlodlo. "Ensemble-based multi-filter feature selection method for DDoS detection in cloud computing." In: *EURASIP Journal on Wireless Communications and Networking* 2016.1 (2016), pp. 1–10. DOI: 10.1186/s13638-016-0623-3.

[131] Yazan Otoum and Amiya Nayak. "As-ids: Anomaly and signature based ids for the internet of things." In: *Journal of Network and Systems Management* 29.3 (2021), pp. 1–26. DOI: 10.1007/s10922-021-09589-6.

[132] Merve Ozkan-Okay, Refik Samet, Ömer Aslan, and Deepti Gupta. "A Comprehensive Systematic Literature Review on Intrusion Detection Systems." In: *Journal of IEEE Access* 9 (2021), pp. 157727–157760. DOI: 10.1109/ACCESS.2021.3129336.

[133] Panos Panagiotou, Notis Mengidis, Theodora Tsikrika, Stefanos Vrochidis, and Ioannis Kompatsiaris. “Host-based intrusion detection using signature-based and ai-driven anomaly detection methods.” In: *Information & Security: An International Journal* 50.1 (2021), pp. 37–48. DOI: 10.11610/isij.5016.

[134] Animesh Patcha and Jung-Min Park. “An overview of anomaly detection techniques: Existing solutions and latest technological trends.” In: *Journal of Computer Networks* 51.12 (2007), pp. 3448–3470. DOI: 10.1016/j.comnet.2007.02.001.

[135] Minh Pham and Doan B Hoang. “SDN applications - The intent-based Northbound Interface realisation for extended applications.” In: *IEEE NetSoft Conference and Workshops (NetSoft)*. 2016, pp. 372–377. DOI: 10.1109/NETSOFT.2016.7502469.

[136] Ngoc Tu Pham, Ernest Foo, Suriadi Suriadi, Helen Jeffrey, and Hassan Fareed M Lahza. “Improving performance of intrusion detection system using ensemble methods and feature selection.” In: *Proceedings of the Australasian Computer Science Week Multiconference*. Association for Computing Machinery, 2018. DOI: 10.1145/3167918.3167951.

[137] James Press and Sandra Wilson. “Choosing between Logistic Regression and Discriminant Analysis.” In: *Journal of the American Statistical Association* 73.364 (1978), pp. 699–705. DOI: 10.1080/01621459.1978.10480080.

[138] Madhukrishna Priyadarsini and Padmalochan Bera. “Software defined networking architecture, traffic management, security, and placement: A survey.” In: *Journal of Computer Networks* 192 (2021), p. 108047. DOI: 10.1016/j.comnet.2021.108047.

[139] Nima Rafiee, Rahil Gholamipoor, Nikolas Adaloglou, Simon Jaxy, Julius Ramakers, and Markus Kollmann. “Self-supervised Anomaly Detection by Self-distillation and Negative Sampling.” In: *Artificial Neural Networks and Machine Learning – ICANN 2022*. Springer Nature Switzerland, 2022, pp. 459–470. DOI: 10.1007/978-3-031-15937-4_39.

[140] Cornelis van Rijsbergen. *Information Retrieval*. 2nd. USA: Butterworth-Heinemann, 1979. DOI: 10.1002/asi.4630300621.

[141] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. “A survey of network-based intrusion detection data sets.” In: *Journal of Computers & Security* 86 (2019), pp. 147–167. DOI: 10.1016/j.cose.2019.06.005.

[142] David Everett Rumelhart and James Lloyd McClelland. “Learning Internal Representations by Error Propagation.” In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press, 1987, pp. 318–362. DOI: 10.7551/mitpress/4943.003.0128.

[143] Aqeel Sahi, David Lai, Yan Li, and Mohammed Diykh. "An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Environment." In: *Journal of IEEE Access* 5 (2017), pp. 6036–6048. DOI: 10.1109/ACCESS.2017.2688460.

[144] Rakesh Salam and Ansuman Bhattacharya. "Efficient greedy heuristic approach for fault-tolerant distributed controller placement in scalable SDN architecture." In: *Journal of Cluster Computing* 25.6 (2022), pp. 4543–4572. DOI: 10.1007/s10586-022-03694-0.

[145] Iqbal Sarker. "Machine learning: Algorithms, real-world applications and research directions." In: *Journal of SN computer science* 2.3 (2021), p. 160. DOI: 10.1007/s42979-021-00592-x.

[146] Patrick Schober, Christa Boer, and Lothar A Schwarte. "Correlation coefficients: appropriate use and interpretation." In: *Journal of Anesthesia & analgesia* 126.5 (2018), pp. 1763–1768. DOI: 10.1213/ANE.0000000000002864.

[147] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. "Sdn Security: A Survey." In: *IEEE SDN for Future Networks and Services (SDN4FNS)*. 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[148] Pratap Chandra Sen, Mahimarnab Hajra, and Mitadru Ghosh. "Supervised Classification Algorithms in Machine Learning: A Survey and Review." In: *Emerging Technology in Modelling and Graphics*. Springer Singapore, 2020, pp. 99–111. DOI: 10.1007/978-981-13-7403-6_11.

[149] Reehan Ali Shah, Yuntao Qian, Dileep Kumar, Munwar Ali, and Muhammad Bux Alvi. "Network Intrusion Detection through Discriminative Feature Selection by Using Sparse Logistic Regression." In: *Journal of Future Internet* 9.4 (2017). DOI: 10.3390/fi9040081.

[150] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization." In: *International Conference on Information Systems Security and Privacy*. Vol. 1. 2018, pp. 108–116. DOI: 10.5220/0006639801080116.

[151] Neha Sharma and Narendra Singh Yadav. "Ensemble Learning based Classification of UNSW-NB15 dataset using Exploratory Data Analysis." In: *9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 2021, pp. 1–7. DOI: 10.1109/ICRITO51393.2021.9596213.

[152] Chris Simmons, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, and Chase Wu. "AVOIDIT: A Cyber Attack Taxonomy." In: *CTIT technical reports series* (2009). URL: https://api.semanticscholar.org/CorpusID:349528 (Last accessed on April 10, 2024).

[153] Maninder Pal Singh and Abhinav Bhandari. "New-flow based DDoS attacks in SDN: Taxonomy, rationales, and research challenges." In: *Journal of Computer Communications* 154 (2020), pp. 509–527. DOI: 10.1016/j.comcom.2020.02.085.

[154] Preeti Sinha, Vijay Kumar Jha, Amit Kumar Rai, and Bharat Bhushan. "Security vulnerabilities, attacks and countermeasures in wireless sensor networks at various layers of OSI reference model: A survey." In: *International Conference on Signal Processing and Communication (ICSPC)*. 2017, pp. 288–293. DOI: 10.1109/CSPC.2017.8305855.

[155] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." In: *Journal of Information Processing & Management* 45.4 (2009), pp. 427–437. DOI: 10.1016/j.ipm.2009.03.002.

[156] Chetan Srinidhi, Seung Wook Kim, Fu-Der Chen, and Anne Martel. "Self-supervised driven consistency training for annotation efficient histopathology image analysis." In: *Journal of Medical Image Analysis* 75 (2022), p. 102256. DOI: 10.1016/j.media.2021.102256.

[157] Rodrigo Vieira Steiner and Emil Lupu. "Towards more practical software-based attestation." In: *Journal of Computer Networks* 149 (2019), pp. 43–55. DOI: 10.1016/j.comnet.2018.11.003.

[158] Jill Stoltzfus. "Logistic Regression: A Brief Primer." In: *Journal of Academic Emergency Medicine* 18.10 (2011), pp. 1099–1104. DOI: 10.1111/j.1553-2712.2011.01185.x.

[159] Tala Talaei Khoei and Naima Kaabouch. "Machine Learning: Models, Challenges, and Research Directions." In: *Journal of Future Internet* 15.10 (2023). DOI: 10.3390/fi15100332.

[160] Jiliang Tang, Salem Alelyani, and Huan Liu. "Feature selection for classification: A review." In: *Data Classification*. CRC Press, 2014, pp. 37–64. DOI: 10.1201/b17320.

[161] Zhulin Tao, Xiaohao Liu, Yewei Xia, Xiang Wang, Lifang Yang, Xianglin Huang, and Tat-Seng Chua. "Self-Supervised Learning for Multimedia Recommendation." In: *IEEE Transactions on Multimedia* 25 (2023), pp. 5107–5116. DOI: 10.1109/TMM.2022.3187556.

[162] Ankit Thakkar and Ritika Lohiya. "A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions." In: *Journal of Artificial Intelligence Review* 55.1 (2022), pp. 453–563. DOI: 10.1007/s10462-021-10037-9.

[163] Talip Ucar, Ehsan Hajiramezanali, and Lindsay Edwards. "SubTab: Subsetting Features of Tabular Data for Self-Supervised Representation Learning." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 18853–18865. DOI: 10.48550/arXiv.2110.04361.

[164] Muhammad Usama, Junaid Qadir, Aunn Raza, Hunain Arif, Kok-lim Alvin Yau, Yehia Elkhatib, Amir Hussain, and Ala Al-Fuqaha. "Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges." In: *Journal of IEEE Access* 7 (2019), pp. 65579–65615. DOI: 10.1109/ACCESS.2019.2916648.

[165] Jyothsna Veeramreddy, Rama Prasad, and Koneti Munivara Prasad. "A review of anomaly based intrusion detection systems." In: *International Journal of Computer Applications* 28.7 (2011), pp. 26–35. DOI: 10.5120/3399-4730.

[166] Yasi Wang, Hongxun Yao, and Sicheng Zhao. "Auto-encoder based dimensionality reduction." In: *Journal of Neurocomputing* 184 (2016), pp. 232–242. DOI: 10.1016/j.neucom.2015.08.104.

[167] Zhendong Wang, Zeyu Li, Junling Wang, and Dahai Li. "Network intrusion detection model based on improved BYOL self-supervised learning." In: *Security and Communication Networks Journal* 2021 (2021), pp. 1–23. DOI: 10.1155/2021/9486949.

[168] Sharyar Wani, Mohammed Imthiyas, Hamad Almohamedh, Khalid M Alhamed, Sultan Almotairi, and Yonis Gulzar. "Distributed Denial of Service (DDoS) Mitigation Using Blockchain—A Comprehensive Insight." In: *Journal of Symmetry* 13.2 (2021). DOI: 10.3390/sym13020227.

[169] Hung-Chuan Wei, Yung-Hao Tung, and Chia-Mu Yu. "Counteracting UDP flooding attacks in SDN." In: *IEEE NetSoft Conference and Workshops (NetSoft)*. 2016, pp. 367–371. DOI: 10.1109/NETSOFT.2016.7502468.

[170] Zhichao Wu, Xin Yang, Xiaopeng Wei, Peijun Yuan, Yuanping Zhang, and Jianming Bai. "A self-supervised anomaly detection algorithm with interpretability." In: *Journal of Expert Systems with Applications* 237 (2024), p. 121539. DOI: 10.1016/j.eswa.2023.121539.

[171] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. "Programmable Switches for in-Networking Classification." In: *IEEE Conference on Computer Communications (IEEE INFOCOM)*. 2021, pp. 1–10. DOI: 10.1109/INFOCOM42981.2021.9488840.

[172] Junfeng Xie, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges." In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 393–430. DOI: 10.1109/COMST.2018.2866942.

[173] Zhaoqi Xiong and Noa Zilberman. "Do Switches Dream of Machine Learning? Toward In-Network Classification." In: *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. Association for Computing Machinery, 2019, pp. 25–33. DOI: 10.1145/3365609.3365864.

[174] Zhen Yang, Xiaodong Liu, Tong Li, Di Wu, Jinjiang Wang, Yunwei Zhao, and Han Han. "A systematic literature review of methods and datasets for anomaly-based network intrusion detection." In: *Journal of Computers & Security* 116 (2022), p. 102675. DOI: 10.1016/j.cose.2022.102675.

[175] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. "Autoencoder-based feature learning for cyber security applications." In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 3854–3861. DOI: 10.1109/IJCNN.2017.7966342.

[176] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. "Self-Supervised Learning for Recommender Systems: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 36.1 (2024), pp. 335–355. DOI: 10.1109/TKDE.2023.3282907.

[177] Yawei Yue, Xingshu Chen, Zhenhui Han, Xuemei Zeng, and Yi Zhu. "Contrastive Learning Enhanced Intrusion Detection." In: *IEEE Transactions on Network and Service Management* 19.4 (2022), pp. 4232–4247. DOI: 10.1109/TNSM.2022.3218843.

[178] Rizgar Zebari, Adnan Abdulazeez, Diyar Zeebaree, Dilovan Zebari, and Jwan Saeed. "A Comprehensive Review of Dimensionality Reduction Techniques for Feature Selection and Feature Extraction." In: *Journal of Applied Science and Technology Trends* 1.1 (2020), pp. 56–70. DOI: 10.38094/jastt1224.

[179] Chunrui Zhang, Gang Wang, Shen Wang, Dechen Zhan, and Mingyong Yin. "Cross-domain network attack detection enabled by heterogeneous transfer learning." In: *Journal of Computer Networks* 227 (2023), p. 109692. DOI: 10.1016/j.comnet.2023.109692.

[180] Zonghua Zhang and Hong Shen. "Application of online-training SVMs for real-time intrusion detection with different considerations." In: *Journal of Computer Communications* 28.12 (2005), pp. 1428–1442. DOI: 10.1016/j.comcom.2005.01.014.

[181] Chengbo Zhou, Christoph Gärtner, Amr Rizk, Boris Koldehofe, Björn Scheuermann, and Ralf Kundel. "RDA: Residence Delay Aggregation for Time-Sensitive Networking." In: *Proceedings of 2024 IEEE Network Operations and Management Symposium (NOMS 2024)*. 2024. DOI: 10.1109/NOMS59830.2024.10574998.

[182] Yuyang Zhou, Guang Cheng, Shanqing Jiang, and Mian Dai. "Building an efficient intrusion detection system based on feature selection and ensemble classifier." In: *Journal of Computer Networks* 174 (2020), p. 107247. DOI: 10.1016/j.comnet.2020.107247.

[183] Noa Zilberman, Yury Audzevich, G. Adam Covington, and Andrew W. Moore. "NetFPGA SUME: Toward 100 Gbps as Research Commodity." In: *Journal of IEEE Micro* 34.5 (2014), pp. 32–41. DOI: 10.1109/MM.2014.61.

All web pages cited in this work have been checked in April 2024. However, due to the dynamic nature of the World Wide Web, their long-term availability cannot be guaranteed.

A APPENDIX

A.1 EXTRACTED FLOW FEATURES FROM NFSTREAM

The core features, post-mortem features, and early statistical features extracted by NFStream and their short descriptions are listed in Table 19, Table 20 and Table 21 below.

Table 19: Core features extracted by NFStream.

No	Feature Name	Description
1	id	Flow identifier.
2	expiration_id	Identifier of flow expiration trigger. Can be 0 for idle_timeout, 1 for active_timeout or -1 for custom expiration.
3	src_ip	Source IP address string representation.
4	src_mac	Source MAC address string representation.
5	src_oui	Source Organizationally Unique Identifier string representation.
6	src_port	Transport layer source port.
7	dst_ip	Destination IP address string representation.
8	dst_mac	Destination MAC address string representation.
9	dst_oui	Destination Organizationally Unique Identifier string representation.
10	dst_port	Transport layer destination port.
11	protocol	Transport layer protocol.
12	ip_version	IP version.
13	vlan_id	Virtual LAN identifier.
14	tunnel_id	Tunnel identifier.
15	protocol_1	Transport layer protocol (ICMP).
16	protocol_2	Transport layer protocol (IGMP).
17	protocol_6	Transport layer protocol (TCP).
18	protocol_17	Transport layer protocol (UDP).
19	protocol_58	Transport layer protocol (ICMPv6).
20	protocol_132	Transport layer protocol (SCTP).

Table 19: Core features extracted by NFStream.

No	Feature Name	Description
21	bidirectional_first_seen_ms	Timestamp in milliseconds on the first flow bidirectional packet. (microseconds in our work)
22	bidirectional_last_seen_ms	Timestamp in milliseconds on the last flow bidirectional packet.
23	bidirectional_duration_ms	Flow bidirectional duration in milliseconds.
24	bidirectional_packets	Flow bidirectional packets accumulator.
25	bidirectional_bytes	Flow bidirectional bytes accumulator (payload in our work).
26	src2dst_first_seen_ms	Timestamp in milliseconds on the first flow src2dst packet.
27	src2dst_last_seen_ms	Timestamp in milliseconds on the last flow src2dst packet.
28	src2dst_duration_ms	Flow src2dst duration in milliseconds.
29	src2dst_packets	Flow src2dst packets accumulator.
30	src2dst_bytes	Flow src2dst bytes accumulator (payload in our work).
31	dst2src_first_seen_ms	Timestamp in milliseconds on the first flow dst2src packet.
32	dst2src_last_seen_ms	Timestamp in milliseconds on the last flow dst2src packet.
33	dst2src_duration_ms	Flow dst2src duration in milliseconds.
34	dst2src_packets	Flow dst2src packets accumulator.
35	dst2src_bytes	Flow dst2src bytes accumulator (payload in our work).

Table 20: Post-mortem features extracted by NFStream.

No	Feature Name	Description
1	analysis_time	Time in seconds during which flows were analyzed.
2	nb_expired_flows	Number of expired flows during the analysis.
3	nb_invalid_checksum	Number of packets with invalid checksum.
4	nb_invalid_ttl	Number of packets with invalid Time-to-Live (TTL).
5	nb_misplaced_packet	Number of misplaced packets in the buffer.
6	nb_flow_mismatch	Number of flows with mismatched bidirectional counters.
7	nb_unhandled_protocols	Number of unhandled transport layer protocols.
8	nb_unhandled_icmp_type	Number of unhandled ICMP types.
9	nb_icmp_error_messages	Number of ICMP error messages.
10	nb_invalid_tcp_flags	Number of packets with invalid TCP flags.
11	nb_tcp_segment_ out_of_order	Number of TCP segments out of order.
12	nb_tcp_segment_ unsynchronized	Number of unsynchronized TCP segments.
13	nb_tcp_segment_not_ captured	Number of TCP segments not captured.
14	nb_missing_tcp_data	Number of missing TCP data segments.
15	nb_fragmented_packets	Number of fragmented packets.
16	nb_fragment_errors	Number of fragment errors.
17	nb_ignored_packets	Number of ignored packets.
18	nb_flow_started	Number of started flows.
19	nb_flow_finished	Number of finished flows.
20	nb_flow_timeout	Number of flows that timed out.
21	nb_ignored_unfavored_ packets	Number of ignored unfavored packets.
22	nb_no_flow_start_packet	Number of packets not starting a flow.
23	nb_flow_start_packet_ ignored	Number of ignored flow start packets.
24	nb_flow_start_packet_ outside_flow	Number of flow start packets outside of a flow.
25	nb_packet_with_flow_ start_tag	Number of packets with the flow start tag.

Table 21: Early statistical features extracted by NFStream.

No	Feature Name	Description
1	splt_piat_ms_1	Inter arrival time, always 0 for the first packet (1 in our work).
2	splt_piat_ms_2	Inter arrival time between 1st and 2nd packets.
3	splt_piat_ms_3	Inter arrival time between 2nd and 3rd packets.
4	splt_piat_ms_4	Inter arrival time between 3rd and 4th packets.
5	splt_piat_ms_5	Inter arrival time between 4th and 5th packets.
6	splt_piat_ms_6	Inter arrival time between 5th and 6th packets.
7	splt_piat_ms_7	Inter arrival time between 6th and 7th packets.
8	splt_piat_ms_8	Inter arrival time between 7th and 8th packets.
9	splt_ps_1	Packet size of 1st packet (payload in our work).
10	splt_ps_2	Packet size of 2nd packet (payload in our work).
11	splt_ps_3	Packet size of 3rd packet (payload in our work).
12	splt_ps_4	Packet size of 4th packet (payload in our work).
13	splt_ps_5	Packet size of 5th packet (payload in our work).
14	splt_ps_6	Packet size of 6th packet (payload in our work).
15	splt_ps_7	Packet size of 7th packet (payload in our work).
16	splt_ps_8	Packet size of 8th packet (payload in our work).
17	splt_direction_1_0	No 1st packet.
18	splt_direction_1_1	1st packet direction (src2dst).
19	splt_direction_1_2	1st packet direction (dst2src).
20	splt_direction_2_0	No 2nd packet.
21	splt_direction_2_1	2nd packet direction (src2dst).
22	splt_direction_2_2	2nd packet direction (dst2src).
23	splt_direction_3_0	No 3rd packet.
24	splt_direction_3_1	3rd packet direction (src2dst).
25	splt_direction_3_2	3rd packet direction (dst2src).
26	splt_direction_4_0	No 4th packet.

Table 21: Early statistical features extracted by NFStream.

No	Feature Name	Description
27	splt_direction_4_1	4th packet direction (src2dst).
28	splt_direction_4_2	4th packet direction (dst2src).
29	splt_direction_5_0	No 5th packet.
30	splt_direction_5_1	5th packet direction (src2dst).
31	splt_direction_5_2	5th packet direction (dst2src).
32	splt_direction_6_0	No 6th packet.
33	splt_direction_6_1	6th packet direction (src2dst).
34	splt_direction_6_2	6th packet direction (dst2src).
35	splt_direction_7_0	No 7th packet.
36	splt_direction_7_1	7th packet direction (src2dst).
37	splt_direction_7_2	7th packet direction (dst2src).
38	splt_direction_8_0	No 8th packet.
39	splt_direction_8_1	8th packet direction (src2dst).
40	splt_direction_8_2	8th packet direction (dst2src).

A.2 EXPLANATION OF THE UTILIZED NETWORK TRAFFIC DATASETS

The following briefly describes the available network traffic datasets used in this thesis for training ML models and evaluating proposed methods.

CICIDS17 Dataset [150]

The CICIDS2017 (Canadian Institute for Cybersecurity Intrusion Detection Systems 2017) dataset is a network traffic data collection designed to evaluate intrusion detection systems. Within this dataset, benign traffic was captured by the abstract behavioral profiles of 25 users through protocols such as HTTP, HTTPS, FTP, SSH, and email. In addition to benign traffic, the dataset includes various types of attack traffic, such as BruteForce, DoS/DDoS, and Botnet attacks. We annotated the network traffic datasets to evaluate the proposed models and retrieve information about the source and destination IP addresses of victims and attackers from the official document of CICIDS17. The attacker and victim IP addresses are demonstrated in Table 22.

Table 22: Pair of IP addresses of attack flows. (* means wildcard)

Dataset	Source IP Address	Destination IP Address
BruteForce	172.16.0.1	*
DoS/DDoS	172.16.0.1	*
Botnet	192.168.10.12	52.6.13.28
	192.168.10.50	172.16.0.1
	172.16.0.1	192.168.10.50
	192.168.10.17	52.7.235.158
	192.168.10.8	205.174.165.73
	192.168.10.5	205.174.165.73
	192.168.10.14	205.174.165.73
	192.168.10.9	205.174.165.73
	205.174.165.73	192.168.10.8
	192.168.10.15	205.174.165.73

UNSW-NB Dataset [122]

The UNSW-NB15 (University of New South Wales - Network-Based 15) is a publicly available dataset for network intrusion detection research. The IXIA PerfectStorm tool was conducted to generate benign and attack traffic using three virtual servers. Therefore, The dataset consists of benign activities and synthetic attack behaviors collected in a controlled environment. The attacker and victim IP addresses are demonstrated in Table 23.

Table 23: Pair of IP addresses of attack flows. (* means wildcard)

Dataset	Source IP Address	Destination IP Address
Attacks	175.45.176.0	149.171.126.10, 149.171.126.11, 149.171.126.12
	175.45.176.1	149.171.126.13, 149.171.126.14, 149.171.126.15
	175.45.176.2	149.171.126.16, 149.171.126.17, 149.171.126.18
	175.45.176.3	149.171.126.19

***CTU-13 Dataset** [52]*

The CTU-13 dataset (CTU University, Czech Republic) is designed to assess Bot-Net detection systems. It consists of 13 scenarios of botnet, benign, and background data, replicating the proportional distribution observed in authentic network settings where attacks constitute a minor fraction. Benign traffic was extracted from university routers to represent real users' behavior. Each of the 13 botnet scenarios is constructed to represent various malware behavior. The attacker and victim IP addresses are demonstrated in Table 24.

Table 24: Pair of IP addresses of attack flows. (* means wildcard)

Dataset	Source IP Address	Destination IP Address
Botnet	*	147.32.84.165
	*	147.32.84.191
	*	147.32.84.192
	*	147.32.84.193
	*	147.32.84.204
	*	147.32.84.205
	*	147.32.84.206
	*	147.32.84.207
	*	147.32.84.208

***CICDoS Dataset** [82]*

CIC-DoS dataset (Canadian Institute for Cybersecurity) is specifically designed to detect slow-rate DoS attacks falling under the MSA category. These attacks were combined with another dataset (ICSX 2012), which consists of high-rate DoS attacks. The attacker and victim IP addresses are demonstrated in Table 25.

***Botnet Dataset**[16]*

Botnet Dataset (Canadian Institute for Cybersecurity) is designed with a focus on generality, realism, and representativeness. In pursuit of these objectives, the dataset

Table 25: Pair of IP addresses of attack flows. (* means wildcard)

Dataset	Source IP Address	Destination IP Address
Attacks	192.168.5.122	192.168.1.101
	192.168.5.122	192.168.4.121
	192.168.5.122	192.168.3.115
	192.168.5.122	192.168.3.114

incorporated diverse centralized and decentralized botnets utilizing various protocols. It comprises 16 botnets characterized by varying lifespans, accommodating both short and long-lived instances to enhance realism. This temporal diversity facilitates the observation of dormant bot functionalities. Moreover, the dataset integrates multiple existing datasets containing real benign flows, thereby providing heterogeneity and a real-world complexity dataset. Below is a list of IP addresses annotated as attack flows, sourced from the official Botnet dataset document. The attacker and victim IP addresses are demonstrated in Table 26.

Table 26: Pair of IP addresses of attack flows. (* means wildcard)

Dataset	Source IP Address	Destination IP Address
Botnet	192.168.4.118	192.168.1.103
	192.168.2.112	131.202.243.84
	192.168.5.122	198.164.30.2
	192.168.2.110	192.168.5.122
	192.168.248.165	*
	74.78.117.238	*
	131.202.243.84	*
	147.32.84.130	*
	192.168.3.65	*

A.3 FLOW FEATURES LIST

In Table 27, a subset of flow features is listed, which are utilized in various analyses, including Exploratory Data Analysis (EDA), Ensemble Feature Selection Method (EFS), and Self-Supervised Contrastive Learning (SSCL) approaches. This subset represents a selection from the complete list of features presented in Section A.1.

Table 27: Utilized flow features for data analysis in EDA and training ML models in the proposed EFS and the proposed SSCL.

ID	Name	ID	Name
1	bidirectional_duration_ms	24	bidirectional_ece_packets
2	bidirectional_packets	25	bidirectional_urg_packets
3	bidirectional_bytes	26	bidirectional_ack_packets
4	src2dst_duration_ms	27	bidirectional_psh_packets
5	src2dst_packets	28	bidirectional_rst_packets
6	src2dst_bytes	29	bidirectional_fin_packets
7	dst2src_duration_ms	30	src2dst_syn_packets
8	dst2src_packets	31	src2dst_cwr_packets
9	dst2src_bytes	32	src2dst_ece_packets
10	bidirectional_min_ps	33	src2dst_urg_packets
11	bidirectional_mean_ps	34	src2dst_ack_packets
12	bidirectional_stddev_ps	35	src2dst_psh_packets
13	bidirectional_max_ps	36	src2dst_rst_packets
14	src2dst_min_ps	37	src2dst_fin_packets
15	src2dst_mean_ps	38	dst2src_syn_packets
16	src2dst_stddev_ps	39	dst2src_cwr_packets
17	src2dst_max_ps	40	dst2src_ece_packets
18	dst2src_min_ps	41	dst2src_urg_packets
19	dst2src_mean_ps	42	dst2src_ack_packets
20	dst2src_stddev_ps	43	dst2src_psh_packets
21	dst2src_max_ps	44	dst2src_rst_packets
22	bidirectional_syn_packets	45	dst2src_fin_packets
23	bidirectional_cwr_packets		

A.4 FLOW FEATURES FOR IN-NETWORK TRAINING

In Table 28, a subset of flow features is listed, which are utilized for training the random forest model deployed in the data plane (DP-IDS). This subset represents a selection from the complete list of features presented in Section A.1.

Table 28: Utilized Features for training the random forest model deployed in the SDN data plane. The definition for these features is available in Section A.1.

ID	Name	ID	Name
1	bidirectional_bytes	11	bidirectional_fin_packets
2	src2dst_packets	12	src2dst_psh_packets
3	src2dst_bytes	13	src2dst_ack_packets
4	dst2src_packets	14	dst2src_ack_packets
5	dst2src_bytes	15	splt_ps_4
6	bidirectional_mean_ps	16	splt_ps_6
7	bidirectional_max_ps	17	splt_ps_7
8	src2dst_max_ps	18	splt_ps_8
9	dst2src_max_ps	19	splt_direction_7_1
10	bidirectional_psh_packets	20	splt_direction_7_2

A.5 PEARSON CORRELATION BETWEEN FEATURES OF VARIOUS DATASETS

In this section, we present heatmap plots illustrating the Pearson correlation between the flow features for three other network traffic datasets. These visualizations help to distinguish between datasets and highlight variations in attack patterns among them. The white lines in the plots indicate zero-variance features, which means that these features are not informative for training ML models. As shown in Figure 26, the white lines occur for different feature flows within a dataset, and these differences can have an impact on relevant feature selection approaches. Furthermore, it's important to note that the diagonal always shows a Pearson correlation of 1, as it represents the correlation between the same flow feature, which essentially compares a feature to itself.

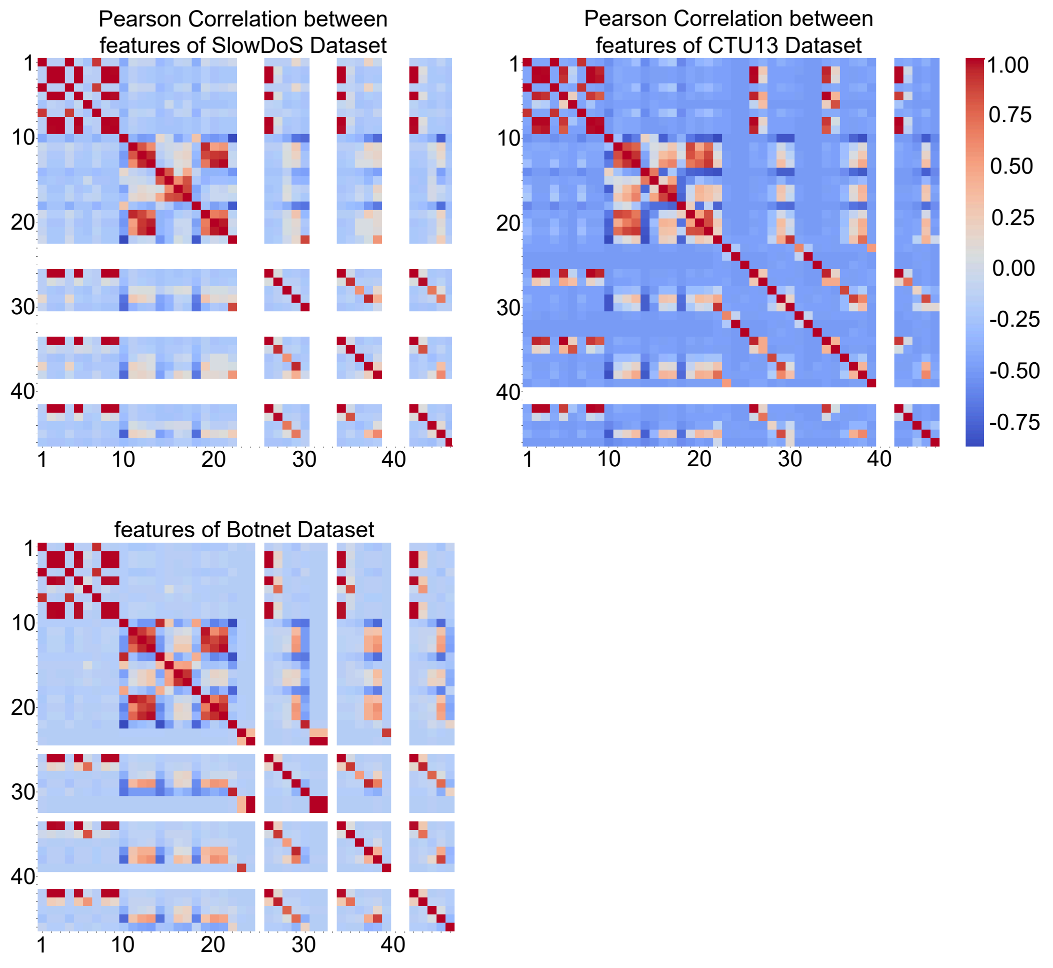

Figure 26: Pearson correlation between flow features within the SlowDoS, CTU13, and Botnet datasets. Red points represent direct linear correlations between features, while blue points indicate indirect correlations.

A.6 IMPACT OF FEATURE SELECTION ON TRAINING TIME

In this section, we present the results of training time using various feature sizes for three additional network traffic datasets: CICIDS17, UNSW-NB, and CICDoS. It's worth noting that Random Forest (RF) and Multi-Layer Perceptron (MLP) are more complex ML models compared to the Logistic Regression (LR) model, which results in longer training times. The results are demonstrated in Table 29.

Table 29: The required training time for each supervised ML model for various feature dimension sizes on three additional network traffic datasets.

Dataset	**Training Time (Second)**			
	5 Features	15 Features	30 Features	45 Features
CICIDS17 with 498,024 flow samples:				
RF	10.01	43.70	94.81	89.47
LR	0.32	2.08	6.42	6.17
MLP	39.05	40.08	42.85	41.41
UNSW-NB with 302,234 flow samples:				
RF	8.44	14.40	25.99	24.06
LR	0.84	1.66	3.15	3.95
MLP	25.39	26.43	28.77	25.76
CICDoS with 275,438 flow samples:				
RF	56.64	57.07	94.73	78.27
LR	0.40	1.37	2.12	3.01
MLP	21.25	22.16	23.22	23.02

A.7 SELECTING FEATURES WITH INDIVIDUAL FEATURE SELECTION APPROACH

In this section, the supervised ML models are trained twice: once on the features extracted solely from the L1-norm LR selector (results in Table 30) and another time on features extracted solely from the L1-norm SVM selector (results in Table 31). The detection performance of each model for each network traffic dataset on various feature dimensions is demonstrated in Table 30 and Table 31. The results in both Table 30 and Table 31 indicate that as the feature dimension increases, the detec-

Table 30: Detection performance of different supervised ML-based NIDS across various network traffic datasets with selected features from the L1-norm logistic regression approach.

Dataset	**F1-Score (%)**				
	5 Features	10 Features	15 Features	20 Features	30 Features
CICIDS17:					
RF	94.6	95.7	96.3	96.3	96.3
LR	94.1	94.3	94.3	91.3	72.4
MLP	94.6	95.9	95.9	96.4	96.7
CICDoS:					
RF	80.2	90.3	96.7	98.7	99.1
LR	75.4	85.8	86.8	92.5	81.6
MLP	80.8	83.2	94.2	94.3	94.9
CTU-13:					
RF	79.8	82.4	90.1	93.7	94.5
LR	94.5	94.5	94.2	94.6	88.7
MLP	84.9	88.1	90.3	95.0	95.5
Botnet:					
RF	68.2	71.7	86.2	90.3	94.3
LR	59.7	63.1	66.9	69.2	72.1
MLP	70.4	71.7	76.5	80.9	83.1
UNSW-NB:					
RF	41.8	89.3	96.8	99.2	99.2
LR	50.6	94.6	97.6	99.4	99.4
MLP	40.8	96.6	99.8	99.8	99.8

tion performance approaches the results achieved from the proposed EFS method. However, in lower feature dimensions, the detection performance tends to be lower. Additionally, results in Table 30 demonstrate that the detection performance of the LR model closely matches the results obtained when utilizing EFS, likely because its structure is similar to the selector's structure. This suggests that the features selected by the L1-norm LR may be the best fit for the LR model.

Table 31: Detection performance of different supervised ML-based NIDS across various network traffic datasets with selected features from the L1-norm Support Vector Machine approach.

Dataset	**F1-Score (%)**				
	5 Features	10 Features	15 Features	20 Features	30 Features
CICIDS17:					
RF	94.2	95.1	96.0	96.2	96.2
LR	94.4	94.7	94.3	94.3	93.7
MLP	94.3	94.7	95.1	95.4	96.2
CICDoS:					
RF	72.1	88.7	93.2	98.3	98.6
LR	70.7	82.5	83.7	80.4	82.4
MLP	75.8	89.0	90.1	94.5	95.0
CTU-13:					
RF	86.1	90.2	95.6	96.1	96.3
LR	83.3	89.5	92.0	93.2	88.8
MLP	86.9	92.1	95.3	95.7	97.8
Botnet:					
RF	67.5	71.8	86.3	93.3	94.2
LR	56.8	59.9	61.4	64.8	66.5
MLP	69.3	71.0	80.3	78.0	83.0
UNSW-NB:					
RF	86.2	93.3	96.6	98.8	99.1
LR	85.1	91.1	94.1	88.6	74.6
MLP	86.1	95.7	97.8	99.0	99.0

A.8 LIST OF ACRONYMS

DDoS	Distributed Denial of Service
NIDSs	Network Intrusion Detection Systems
ML	Machine Learning
SDN	Software-Defined Networking
PCAP	Packet Capture
OSI	Open Systems Interconnection
MAC	Media Access Control
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ICMP	Internet Control Message Protocol
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
ALOHA	Additive Link On-line Hawaii Area
LR	Logistic Regression
RF	Random Forest
DT	Decision Tree
MLP	Multi-Layer Perceptron
XGBoost	Extreme Gradient Boosting
AE	Auto Encoder
SSL	Self-Supervised Learning
GNN	Graph Neural Network
EDA	Exploratory Data Analysis
EFS	Ensemble Feature Selection
DD-EFS	Data-Driven Ensemble Feature Selection
QoS	Quality of Service
NLP	Natural Language Processing
NT-Xent	Normalized Temperature-Scaled Cross-Entropy
SSCL	Self-Supervised Contrastive Learning
AUROC	Area Under the Receiver Operating Characteristic
LLM	Large Language Model
t-SNE	t-Distributed Stochastic Neighbor Embedding
MSA	Multi-Stage Attacks

IDS	Intrusion Detection System
CML-IDS	Collaborative Machine Learning-based Intrusion Detection System
P4	Programming Protocol-Independent Packet Processors
CP	Control Plane
CP-IDS	Control Plane Intrusion Detection System
DP	Data Plane
DP-IDS	Data Plane Intrusion Detection System
MA	Match-Action
DoS	Denial of Service

B SUPERVISED STUDENT THESES

[1] Hengyu Liu. "Incorporating Collaborative Online Machine Learning for Intrusion Detection in Software Defined Networking." Master Thesis. KOM-M-0775. TU Darmstadt, 2024.

[2] Kexin Wang. "Generating Adversarial C2 Communication Attacks leveraging the Shift Distribution Strategy." Master Thesis. KOM-M-0773. TU Darmstadt, 2024.

[3] Chengbo Zhou. "A Collaborative Machine Learning-based Network Intrusion Detection Architecture in Software Defined Networking." Master Thesis. KOM-M-0784. TU Darmstadt, 2023. Received the Best Master Thesis Award at the Multimedia Communications Lab.

[4] Yizi Liu. "In-Network intrusion detection system leveraging decision tree model inference." Master Thesis. KOM-M-0752. TU Darmstadt, 2023.

[5] Leonard Anderweit. "In-Network DDoS Attack Detection Leveraging Time Series Data in Programmable Data Planes." Bachelor Thesis. KOM-B-0713. TU Darmstadt, 2022.

[6] Stefan Stegmueller. "Generative Adversarial Network-based Robustness Evaluation of Machine Learning Classification Algorithms for DDoS-Attacks." Master Thesis. KOM-M-0737. TU Darmstadt, 2022. Received the Best Master Thesis Award at the Multimedia Communications Lab.

C AUTHOR'S PUBLICATIONS

MAIN PUBLICATIONS

[1] Pegah Golchin, Nima Rafiee, and Ralf Kundel. "A Data-Driven Solution for Improving Transferability of Traffic Flow Feature Selection." In: *2024 IFIP Networking Conference (IFIP Networking)*. IEEE. 2024, pp. 1–3. Forthcoming.

[2] Pegah Golchin, Nima Rafiee, Mehrdad Hajizadeh, Ahmad Khalil, Ralf Kundel, and Ralf Steinmetz. "SSCL-IDS: Enhancing Generalization of Intrusion Detection with Self-Supervised Contrastive Learning." In: *2024 IFIP Networking Conference (IFIP Networking)*. IEEE. 2024, pp. 1–9. Forthcoming.

[3] Pegah Golchin, Chengbo Zhou, Pratyush Agnihotri, Pratyush Agnihotri, Mehrdad Hajizadeh, Ralf Kundel, and Ralf Steinmetz. "CML-IDS: Enhancing Intrusion Detection in SDN Through Collaborative Machine Learning." In: *19th International Conference on Network and Service Management (CNSM)*. 2023, pp. 1–9. DOI: 10.23919/CNSM59352.2023.10327863.

[4] Pegah Golchin, Jannis Weil, Ralf Kundel, and Ralf Steinmetz. "Dynamic network intrusion detection system in Software-Defined Networking." In: *2nd Workshop on Machine Learning & Networking (MaLeNe), co-located with the 5th International Conference on Networked Systems (NetSys 2023)*. 2023, pp. 1–2.

[5] Pegah Golchin, Leonard Anderweit, Julian Zobel, Ralf Kundel, and Ralf Steinmetz. "In-Network SYN Flooding DDoS Attack Detection Utilizing P4 Switches." In: *Proceedings of the 3rd KuVS Fachgespräch "Network Softwarization"*. 2022, pp. 1–2. DOI: 10.15496/publikation-67441.

[6] Pegah Golchin, Ralf Kundel, Tim Steuer, Rhaban Hark, and Ralf Steinmetz. "Improving DDoS Attack Detection Leveraging a Multi-aspect Ensemble Feature Selection." In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022, pp. 1–5. DOI: 10.1109/NOMS54207.2022.9789763.

CO-AUTHORED PUBLICATIONS

[7] Ahmad Khalil, Tizian Dege, Pegah Golchin, Rostyslav Olshevskyi, Antonio Fernandez Anta, and Tobias Meuser. *Federated Learning with Heterogeneous Data Handling for Robust Vehicular Object Detection*. 2024. DOI: 10.48550/arXiv.2405.01108. arXiv: 2405.01108 [cs.CV].

[8] Mehrdad Hajizadeh, Sudip Barua, and Pegah Golchin. "FSA-IDS: A Flow-based Self-Active Intrusion Detection System." In: *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2023, pp. 1–9. doi: `10.1109/NOMS56928.2023.10154343`.

[9] Daniel Mulnaes, Pegah Golchin, Filip Koenig, and Holger Gohlke. "Topdomain: exhaustive protein domain boundary metaprediction combining multi-source information and deep learning." In: *Journal of chemical theory and computation* 17.7 (2021), pp. 4599–4613. doi: `10.1021/acs.jctc.1c00129.`.

[10] Annika Behrendt, Pegah Golchin, Filip König, Daniel Mulnaes, Amelie Stalke, Carola Dröge, Verena Keitel, and Holger Gohlke. "Vasor: Accurate prediction of variant effects for amino acid substitutions in multidrug resistance protein 3." In: *Journal of Hepatology Communications* 6.11 (2022), pp. 3098–3111. doi: `10.1002/hep4.2088`.

D ERKLÄRUNGEN LAUT PROMOTIONSORDNUNG

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass von mir zu keinem vorherigen Zeitpunkt bereits ein Promotionsversuch unternommen wurde. Andernfalls versichere ich, dass der promotionsführende Fachbereich über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs informiert ist.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation, abgesehen von den in ihr ausdrücklich genannten Hilfsmitteln, selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde. Weiterhin versichere ich, dass die "Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Technischen Universität Darmstadt" sowie die "Leitlinien zum Umgang mit digitalen Forschungsdaten an der TU Darmstadt" in den jeweils aktuellen Versionen bei der Verfassung der Dissertation beachtet wurden.

§ 9 Abs. 2 PromO

Ich versichere hiermit, dass die vorliegende Dissertation bisher noch nicht zu Prüfungszwecken gedient hat.

Darmstadt, 7. Mai 2024

Pegah Golchin

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. `classicthesis` is available for both LaTeX and LyX:

`https://bitbucket.org/amiede/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Final Version as of August 14, 2024 (`classicthesis`).