From Flow to Packet: A Unified Machine Learning Approach for Advanced Intrusion Detection

Article *in* Security and Communication Networks · July 2025

DOI: 10.1155/sec/5729035

Didik Sudyana
National Cheng Kung University
31 PUBLICATIONS **163** CITATIONS
SEE PROFILE

Fietyata Yudha
Islamic University of Indonesia
25 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Y.R. Lin
National Yang Ming Chiao Tung University
298 PUBLICATIONS **5,044** CITATIONS
SEE PROFILE

Po-Ching Lin Lin
National Yang Ming Chiao Tung University
58 PUBLICATIONS **926** CITATIONS
SEE PROFILE

Wiley
Security and Communication Networks
Volume 2025, Article ID 5729035, 25 pages
https://doi.org/10.1155/sec/5729035

Research Article

From Flow to Packet: A Unified Machine Learning Approach for Advanced Intrusion Detection

Didik Sudyana,[1] **Fietyata Yudha**,[2] **Ying-Dar Lin**,[2] **Chia-Hung Lai**,[3] **Po-Ching Lin**,[4] **and Ren-Hung Hwang**[5]

[1]*Computer and Network Center, National Cheng Kung University, Tainan 701, Taiwan*
[2]*Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan*
[3]*Industrial Technology Research Institute, Hsinchu 310, Taiwan*
[4]*Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 600, Taiwan*
[5]*College of Artificial Intelligence, National Yang Ming Chiao Tung University, Tainan 710, Taiwan*

Correspondence should be addressed to Ying-Dar Lin; ydlin@cs.nycu.edu.tw

Received 22 June 2024; Accepted 5 June 2025

Academic Editor: Zhiyuan Tan

In the era of advanced networking with 5G integration, the need for efficient and scalable intrusion detection systems has become critical to securing large-scale digital infrastructures. Traditional intrusion detection approaches either analyze individual packets yielding high computational costs or rely solely on flow-based data, which can miss important sequence-level information critical to identifying interservice communications and attack behaviors. To address this, we propose a unified machine learning approach that integrates flow-based and packet-based detection using convolutional neural networks (CNNs) for advanced intrusion detection. Our method prioritizes flow-based detection for short flows as the first defense layer and selectively invokes packet-based detection for longer flows or cases deemed uncertain. Uncertain predictions from the flow-based stage are identified using a confidence threshold and re-evaluated by the packet-based system. We validate our method using a systematically generated dataset from a microservices environment alongside benchmark datasets, including CIC-IDS-2017, CIC-IDS-2018, and CRE-MEv2. This hybrid detection strategy yields strong performance in both accuracy and efficiency. Specifically, our approach reduces the computational cost by up to 24 × (approximately 1.38 orders of magnitude) compared to relying solely on packet-based analysis. Additionally, the model demonstrates strong generalization with detection rates of 95% and 100% for flow- and packet-based detection, respectively, even against previously unseen attacks generated through behavioral variations and command-level perturbations.

Keywords: deep learning; microservices security; ml for intrusion detection; unified flow and packet

1. Introduction

The advent of 5G technology has paved the way for the development of on-demand services that require low latency to function effectively [1]. As a result, network and system architectures have evolved to accommodate these requirements by transitioning to distributed environments. One of the key aspects of this evolution is the adoption of distributed systems, such as microservices. Microservices allow for the decomposition of an application into separate components or "services" that can be managed independently. Container-based applications have become the standard for effectively implementing microservices, with Kubernetes being the most widely used container orchestration platform.

System security and intrusion detection have become increasingly crucial with the rising adoption of 5G and microservices. Addressing these needs involves securing microservice systems and detecting intrusions within them. Securing microservices is challenging for several reasons [2]: (1) They are distributed across nodes with interservice communications, and these nonlinear communication patterns make distinguishing normal from anomalous behavior difficult. With potentially hundreds of concurrently running microservices, such systems are harder to secure than traditional monolithic ones. (2) Unlike monolithic systems where components communicate locally, microservices communicate over networks, enlarging the attack surface. Each service can be an entry for attackers, with the complex communication allowing lateral attacker movement. (3) Microservices communicate using diverse protocols and event-driven architectures, complicating the identification of malicious activities. (4) These intricate patterns can increase false positives (FPs) and negatives (FNs) in intrusion detection.

A prevalent strategy is implementing distributed intrusion detection on each network node, allowing for real-time anomaly identification by analyzing network traffic data. There are two forms of network data that can be used for intrusion detection: packet-based and flow-based.

Packet-based detection focuses on the packet, the fundamental unit of data transmission in a network. Each packet carries detailed information about its source, destination, and content. This method can involve analyzing the payload of packet data [3–5]. However, the prevalence of encryption in modern protocols poses significant challenges to achieving accurate classification [6]. Alternatively, packet-based detection may employ sampling techniques based on specific fields from the 5-tuple source IP, destination IP, source port, destination port, and protocol type [7, 8]. This approach offers granular insights into each packet and tends to improve accuracy. Nevertheless, due to the large volume of packets in modern networks, this method demands significant computational resources to process and analyze the data.

On the other hand, flow-based data collate packets into flows using the same 5-tuple fields, providing a statistical snapshot of traffic between endpoints. This summary might include metrics such as packet count, byte count, and flow duration, facilitating a comprehensive network behavior analysis. While third-party tools can extract features from these flows and calculate statistics, offering a less computationally intensive alternative to packet-by-packet analysis, this method captures only statistical patterns, omitting the nuanced sequences and trajectories indicative of network anomalies. Additionally, the necessity to wait for a flow's completion before calculating the correct statistics limits its applicability to real-time detection. Analyzing incomplete flows leads to lower detection accuracy and higher FP rates, diminishing the method's overall effectiveness [9].

Packet-based and flow-based detection methods each have their strengths and limitations. While packet-based detection ensures high accuracy, it demands significant computational resources. Conversely, flow-based detection is less resource-intensive but experiences reduced accuracy, requiring the completion of extensive flows before detection can take place [10]. A unified approach, merging both methods, could optimally harness their advantages and enhance detection efficiency.

In the past, network data were typically analyzed using signature-based detection. However, this approach proved ineffective due to the prevalence of encrypted packets in modern networks. Furthermore, it was also constrained by its reliance on predefined attack signatures and exhibited a high FN rate, leading to missed unknown attack attacks. Machine learning (ML) has now emerged as a superior alternative by learning the patterns and behaviors of network data. ML models can be trained on network data, enabling them to distinguish between normal and malicious network activity. This approach significantly enhances detection performance while reducing the FN rate [11] and can work effectively even in the presence of encrypted packets [12–14].

Deep learning (DL) algorithms, a subset of ML, have demonstrated impressive effectiveness in intrusion detection. They excel at profiling traffic and identifying malicious activities [15]. One such algorithm, the convolutional neural network (CNN), is frequently deployed due to its capacity to autonomously extract features from data, making it ideally suited for traffic profiling. With CNN, traffic features can be directly learned and profiled, facilitating more accurate intrusion detection even when dealing with encrypted packets [16–18]. This method can be implemented in a unified approach for both flow and packet data, learning patterns and identifying malicious behavior within the data.

The motivation for this work arises from the limitations of existing intrusion detection methods in addressing the challenges posed by modern network environments. Current unified or hybrid approaches predominantly focus on either flow-based or packet-based data, with limited exploration of how these can be combined effectively [19–31]. While [8] introduced a hybrid method that integrates flow and packet features for joint processing, it incurs significant computational overhead due to the simultaneous processing of both data types. In contrast, our work seeks to address these shortcomings by proposing an innovative unified approach that leverages the strengths of both flow- and packet-based detection while minimizing resource demands. Specifically, our method uses flow-based detection as the primary defense for shorter flows, reserving the more resource-intensive packet-based detection for longer flows or cases where flow-based confidence scores fall below a specified threshold. By forwarding uncertain flow-based results for packet-level reanalysis, our approach reduces FPs and FNs, enhances computational efficiency, and enables real-time detection, making it well-suited for securing dynamic and distributed systems such as microservices.

Our work's contributions to the field are fourfold:

- To the best of our knowledge, this study is the first to introduce a novel unified detection framework that combines flow- and packet-based methods, leveraging their respective strengths to improve both accuracy and efficiency.

- We propose a confidence-driven mechanism that flags uncertain flow-based detections for further analysis at the packet level, significantly reducing FPs and FNs.
- We design an approach that prioritizes computational efficiency, utilizing flow-based detection for most traffic and engaging packet-based detection selectively, thereby optimizing resource usage.
- We validate the proposed framework through extensive experiments across multiple datasets, demonstrating its superior performance in terms of detection accuracy, computational efficiency, and robustness against state-of-the-art methods.

Furthermore, in order to enhance our knowledge, we also investigate: (1) the performance of various detection methods, including flow- and packet-based approaches; (2) the influence of different threshold types on grey zone data, which impacts FP and FN rates; (3) the effectiveness of the unified flow-and-packet approach, to evaluate its performance in terms of detection accuracy and computational demand; and (4) unknown attack testing, to assess the effects of unknown attack on the performance of the unified flow-packet approach.

The remaining sections of this paper are structured as follows. Section 2 discusses the related works on unified and hybrid intrusion detection. In Section 3, the problem formulation is defined. Section 4 discusses the design solution, and Section 5 explains the system's implementation. In Section 6, we present the results and discussions. Finally, in Section 7, we draw the conclusions based on our findings.

2. Related Works

We provide a comprehensive overview of previous research in the field of unified or hybrid network anomaly detection and intrusion detection systems (IDSs), which is summarized in Table 1. Although numerous studies have proposed methods for anomaly detection or intrusion detection using ML, our focus is specifically on works that utilize hybrid approaches, whether through mixed models or data. This distinct focus allows us to highlight the unique contributions of hybrid methodologies in enhancing detection capabilities. The table provides a comparative analysis of each study, focusing on several key aspects, such as the employed hybrid detection method, types of input data used for training and testing, the specific ML methods applied, proposed solutions, and the used datasets. Given the multitude of research studies conducted on hybrid intrusion detection in network traffic, each study adopts a distinct definition of hybrid detection. To enhance clarity, we categorize these definitions into three categories.

The first category involves model hybrid, which refers to combining classification methods using binary classification in the first level and multiclass in the second level or employing a combination of DL models. The classifier combination was performed by [19–22] through combining binary and multiclass classification. In their method, binary detection is employed at the first level, while multiclass classification is utilized at the second level. For instance, [19] used a Decision Tree at the first level and a Random Forest (RF) at the second level, training and testing their model on CIC-IDS2017 and UNSW-NB15 datasets, achieving $F1$ scores of 100% and 97%, respectively. [20] employed a multimodal deep autoencoder for binary detection and soft-output classifiers for multiclass classification, improving performance by 5% over the baselines. Similarly, [21] used SparkML at the first level and a Conv-LSTM model at the second level, training and testing on the ISCX-IDS 2012 dataset to achieve a 97.29% accuracy. This approach aligns with the work of [22], which initially employed an autoencoder for anomaly detection at the first level, and then utilized a RF trained exclusively on malicious data at the second level, achieving a balanced accuracy of 96%. It is worth noting that all mentioned datasets are flow-based and rely on statistical features.

Furthermore, several studies have combined DL models to enhance intrusion detection performance. These approaches often involve using DL for feature extraction or selection, followed by training with ML or DL techniques. For instance, [31] employed grey wolf optimization (GWO) for feature selection and trained a CNN model, while [30] extracted features using a CNN model and trained a long short-term memory (LSTM) model. Similarly, [23] utilized a CNN for feature extraction and support vector machine (SVM) for model training. Other works include [26], which employed LSTM and Naive Bayes; [27], which integrated CNN, gated recurrent unit (GRU), and BiLSTM models; [25], which employed a CNN–LSTM combination; and [28], which combined graph convolution network (GCN) and LSTM models. Additionally, [29] applied stacked ensemble learning using RF, CatBoost, and XGBoost as base learners, with a multilayer perceptron (MLP) as the meta-learner. Moreover, some studies adopted ensemble techniques that combine multiple DL models for final decision-making. For example, [24] utilized a Cu-LSTM–GRU architecture.

Most existing hybrid detection approaches discussed above focus on model hybrid techniques, where multiple classifiers are combined at different stages to improve classification performance. However, these approaches do not explicitly optimize the integration of flow- and packet-based detection, instead focusing on refining classification models rather than reducing computational overhead.

Among prior works, MEMBER [8] is the closest competitor to our approach, as it also integrates both flow- and packet-based data. MEMBER employs a multitask learning model that fuses statistical flow-based features and raw packet details into a single CNN model, processing both data types simultaneously. While this approach enhances feature representation, it introduces significant computational overhead because it processes both flow and packet data at all times, even for benign traffic. This redundant computation makes MEMBER less efficient for real-time deployment in large-scale networks, as it lacks a mechanism to selectively process only relevant traffic.

In contrast, our method follows a tiered decision-making approach, where flow-based detection serves as the first defense, and packet-based detection is selectively applied only for uncertain cases in the "grey zone." This confidence-

Table 1: Summary of the related works on unified/hybrid intrusion detection.

Paper	Hybrid detection	ML input		Method	Solution	Dataset
		Flow	Packet			
[19]	Model hybrid	O	X	Binary in 1st level multiclass in 2nd level	Decision tree–random forest	CIC-IDS-2017, UNSW-NB15
[20]					Autoencoder–soft output	Bot-IoT dataset
[21]					SparkML–Conv-LSTM	ISCX-IDS 2012
[22]					Autoencoder–random forest	CIC-IDS-2017
[23]				Combination of deep learning models	CNN as feature extractor, SVM as classifier	CIC-IDS-2017
[24]					Cu-LSTM–GRU	CIC-IDS-2018
[25]					CNN–LSTM	NSL-KDD
[26]					LSTM–Naive Bayes	CIC-DDoS2019, CIC-IoT2023, CICIoV2024
[27]					CNN–GRU–BiLSTM	NSL–KDD
[28]					GCN–LSTM	IoT-23
[29]					RF–CatBoost–XGBoost–MLP	CIC-IDS-2017, UNSW-NB15
[30]		X	O		CNN as feature extractor, LSTM CosMargin layer as classifier	ISCX-IDS 2012 CIC-IDS-2017
[31]					GWO as feature selection, CNN as anomaly classification	DARPA98 KDD99
[8]	Data hybrid	O	O	Fusion statistical and packet features	Multitask learning model with one CNN model	BoT-IoT, CIC-IDS-2017, UNSW-NB51, ISCX-IDS2012
Ours	Model and data hybrid			Unified flow-based and packet-based	Flow-based in 1st line, packet-based in 2nd line	Self-generated (microservices environment), CIC-IDS-2017, CIC-IDS-2018, CREMEv2

driven mechanism ensures that most network traffic is processed efficiently at the flow level, preventing unnecessary packet-based analysis for benign traffic. By escalating only ambiguous cases to packet-based detection, our approach significantly reduces computational costs while maintaining high accuracy.

Furthermore, while MEMBER merges flow and packet features into a unified model, our approach separates flow and packet detection into two distinct CNN models, allowing independent processing at different levels. This separation improves adaptability, ensuring that packet-based detection is only triggered when the flow-based model lacks confidence in its classification. The resulting efficiency gain makes our method better suited for real-time intrusion detection in large-scale and resource-constrained environments.

3. Problem Formulation

This section explains our problem definition and the variables and notations employed, summarized in Table 2. Given the training dataset, ML algorithm, and testing dataset, we need to determine the most accurate ML model and the ideal threshold detection parameters for flow-based and packet-based approaches. Additionally, we must establish the threshold for the grey zone data to filter out low-confidence detection results. Our objective is to maximize the $F1$ score of the unified flow-and-packet method. The problem statement is then formally defined as follows:

Input: The training dataset R^B, ML algorithm K, and testing dataset R^G.

Output: Most accurate ML models with threshold detection for flow-based L^F, τ^F; packet-based L^P, τ^P; and the optimal threshold of grey zone data $\tau_{\min}^{\mathrm{FN}}, \tau_{\max}^{\mathrm{FN}}, \tau_{\min}^{\mathrm{FP}}, \tau_{\max}^{\mathrm{FP}}$.

Objective: Maximize $F1$ score of unified flow-and-packet ML model.

4. System Design

We propose a unified flow-and-packet intrusion detection approach based on ML with the goal of improving the efficacy and efficiency of network intrusion detection while minimizing computational overhead. Our method utilizes flow-based detection as the first-line defender by aggregating packets into flows and extracting features by calculating the statistics of a flow. This process demands less computational resources while providing a confidence score for intrusion detection. If the confidence score is low, packet-based detection will re-check the traffic. In this section, we discuss the details of this unified approach.

4.1. Unified Flow-and-Packet Intrusion Detection Algorithm.

The algorithm runs in a live mode that processes the network traffic. It comprises one top-level stage and four detailed steps, as illustrated in Figure 1. It begins by initializing the ML models and various parameters, followed by an exhaustive search for the grey zone threshold. Next, it starts "Packet Handler" step to handle the incoming traffic in the form of packets. Every incoming packet is then grouped into flow entries. A flow represents a sequence of data packets sharing common characteristics, such as source and destination addresses, ports, and protocol type. Once the packet has been grouped into their appropriate flow entries, the subsequent processes occur in parallel. The first parallel process involves continuously receiving new packets, while the second parallel process gets the total number of flows and examines those flows in the "Flow Checker" step to determine whether they meet the criteria to be processed by the intrusion detection module. If the decision is no, it returns to receive and group the next incoming packets.

There are two criteria for determining when a flow is analyzed by intrusion detection: when it is finished or too long. When the flow is finished, it will be sent to flow-based detection. Two criteria are employed to determine when a flow is considered finished: (1) utilizing a timeout threshold and (2) examining the TCP flags. The first criterion involves setting a predetermined period of inactivity (e.g., 30 s), after which the absence of packets indicates the end of the flow. This criterion is also applied to define the end of UDP and ICMP connections. The second criterion involves examining the TCP flag. If the flag indicates either FIN (Finish) or RST (Reset), the flow is then categorized as a finished flow. On the other hand, when the flow is too long, it will be sent to packet-based detection. We employ a flow duration threshold to ascertain whether a flow has become excessively lengthy. When the flow's duration reaches this threshold, we cease the reception of further incoming packets and proceed directly to the packet-based detection process.

When the flow-based detection processes the flow, it outputs the confidence score pF. The system then checks the confidence score of the flow-based detection in the "Grey Zone Checking" step to determine whether the result belongs to the grey zone data or not. Grey zone data are the condition when the flow-based detection does not have good confidence to decide whether the data are benign traffic or anomaly traffic. If it is grey zone data, then the packet data, which has been grouped from this flow, are sent to "packet-based detection" for further checking. Finally, after the data are processed by either flow-based or packet-based detection, the output of confidence score p_i^F or p_i^P is checked with the defined threshold to classify it as benign or anomaly in "Detection Labeling."

4.2. Grey Zone Threshold.

IDSs have demonstrated success in detecting network attacks, but they often generate numerous false alarms, straining network operators and reducing effectiveness. Even just 1% of FP data might cause hundreds of false alarms every day, which strains the network operator. Combining flow-packet detection aims to reduce FPs and FNs while maintaining low computational overhead. In this context, the positive class represents malicious flows, while the negative class represents benign flows. FP occurs when benign flows are incorrectly flagged as malicious, and FN stands for undetected malicious flows.

Furthermore, to improve the performance of our intrusion detection, we introduce a new term called "grey zone data" to filter low-confidence detection results by our flow-based detection as the first defender. Grey zone data occur

Table 2: Notations table.

Notation	Meaning
R^B	Training dataset
R^G	Testing dataset
S_j	j-th packet data
F_i	i-th flow data
F^*	i-th extracted flow
$\|F\|$	The total number of flows F
K	ML algorithm
L^F	Best ML model for flow-based detection
L^P	Best ML model for packet-based detection
τ^Q	Threshold value for waiting time
τ^S	Threshold value for flow duration
p_i^F	Confidence score of flow-based detection for each i-th flow data
p_i^P	Confidence score of packet-based detection for each i-th flow data
τ^F	Threshold value for flow-based detection
τ^P	Threshold value for packet-based detection
y	The percentile number from 0 to 100 $y = 0, 1, 2100$
p_y^{FN}	Set of percentile-based probabilities for false-negative data $p^{\mathrm{FN}} = \{p^{\mathrm{FN}} \| y \in [0, 100]\}$
p_y^{FP}	Set of percentile-based probabilities for false-positive data $p^{\mathrm{FP}} = \{p^{\mathrm{FP}} \| y \in [0, 100]\}$
p_y^{TN}	Set of percentile-based probabilities for true-negative data $p^{\mathrm{TN}} = \{p^{\mathrm{TN}} \| y \in [0, 100]\}$
p_y^{TP}	Set of percentile-based probabilities for true-positive data $p^{\mathrm{TP}} = \{p^{\mathrm{TP}} \| y \in [0, 100]\}$
$p_{\min}^{\mathrm{FN}}$	The minimum false-negative probability percentile
$p_{\max}^{\mathrm{FN}}$	The maximum false-negative probability percentile
$p_{\min}^{\mathrm{FP}}$	The minimum false-positive probability percentile
$p_{\max}^{\mathrm{FP}}$	The maximum false-positive probability percentile
$\tau_{\min}^{\mathrm{FN}}$	The minimum threshold value for false-negative data
$\tau_{\max}^{\mathrm{FN}}$	The maximum threshold value for false-negative data
$\tau_{\min}^{\mathrm{FP}}$	The minimum threshold value for false-positive data
$\tau_{\max}^{\mathrm{FP}}$	The maximum threshold value for false-positive data

when flow-based detection cannot confidently classify traffic as benign or malicious. We evaluate the confidence score of each flow against a defined threshold and flag any data falling within the threshold range as the grey zone. All the grey zone data are then forwarded to packet-based detection for further analysis.

We used two carefully configured thresholds for FN and FP data to determine grey zone data. Tight thresholds burden resources with excessive correct data checking, while loose thresholds result in many FP or FN data being lost to be rechecked by packet-based detection.

Figure 2 provides an illustration of the defined thresholds. When flow-based detection outputs a high confidence result, the confidence score leans closer to either 0 or 1. However, if the confidence score is far from 0 or 1, it is categorized as a "grey zone," representing a low-confidence result. To manage the computational load, we set a maximum tolerance of only 0.5% for correct data. This tolerance limit is set to reduce the number of correct data entries that the packet-based detection system needs to verify. Based on these considerations, the values for $\tau_{\min}^{\mathrm{FN}}$ and $\tau_{\max}^{\mathrm{FP}}$ can be fine-tuned to optimize the unified system.

Figures 3(a) and 3(b) illustrate the algorithm for finding the grey zone threshold for FNs and FPs, respectively. After running flow-based detection on the testing data, we obtain the detection results in terms of confidence scores. We then group the results into FN, FP, true negative, and true positive by comparing the detection results with the ground truth. Thus, we compute the percentiles of the confidence scores for each group and store them in a set. These percentiles are used to determine a neither too loose nor too tight threshold. We calculate the percentile of the confidence score by using the notation: $k = h/100 \times (n + 1)$, where k represents the value of the confidence score in the dataset that corresponds to the desired percentile, h is the percentile we want to calculate, and n is the total number of confidence scores in the dataset.

For example, to calculate the 90th percentile, denoted as k, from the example set of confidence scores (0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5), we can use the formula mentioned earlier: $k = 90/100 \times (9 + 1)$. This gives $k = 9$. This yields $k = 9$, indicating that the 90th percentile corresponds to the 9th value in the sorted dataset, which in this case is 0.5.

We employ the algorithm in Figure 3(a) to determine the minimum and maximum thresholds for FN data, using probability percentiles for FN (p_y^{FN}) and true-negative (p_y^{FP}) data as inputs. Initially, we assign the $\{0^{\mathrm{th}}$ percentile$\}$ of FN p_0^{FN} as the minimum value $p_{\min}^{\mathrm{FN}}$ and the 100^{th} percentile of FN p_Y^{FN} as the maximum value $p_{\max}^{\mathrm{FN}}$. We then adjust $p_{\min}^{\mathrm{FN}}$ to find an appropriate threshold by checking if it is lower than the 99.5^{th} percentile value of true negative $p_{99.5^{\mathrm{th}}}^{\mathrm{TN}}$. If not, we increment the percentile by 0.1 and update $p_{\min}^{\mathrm{FN}}$. We continue adjusting until $p_{\min}^{\mathrm{FN}}$ is lower than $p_{95^{\mathrm{th}}}^{\mathrm{TN}}$, and accept $p_{\min}^{\mathrm{FN}}$ and $p_{\max}^{\mathrm{FN}}$ as the FN data thresholds, $\tau_{\min}^{\mathrm{FN}}$ and $\tau_{\max}^{\mathrm{FN}}$, respectively.

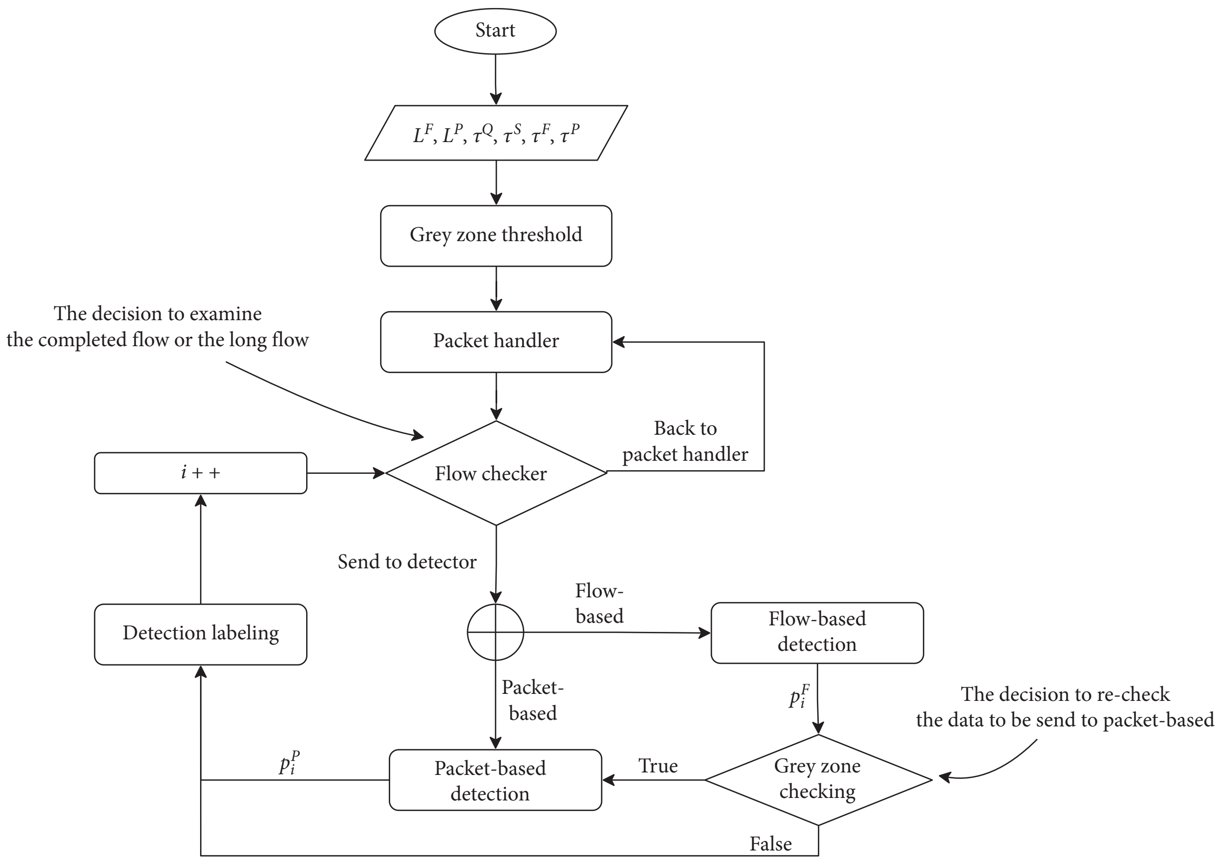


Figure 1: Unified flow-and-packet intrusion detection algorithm.

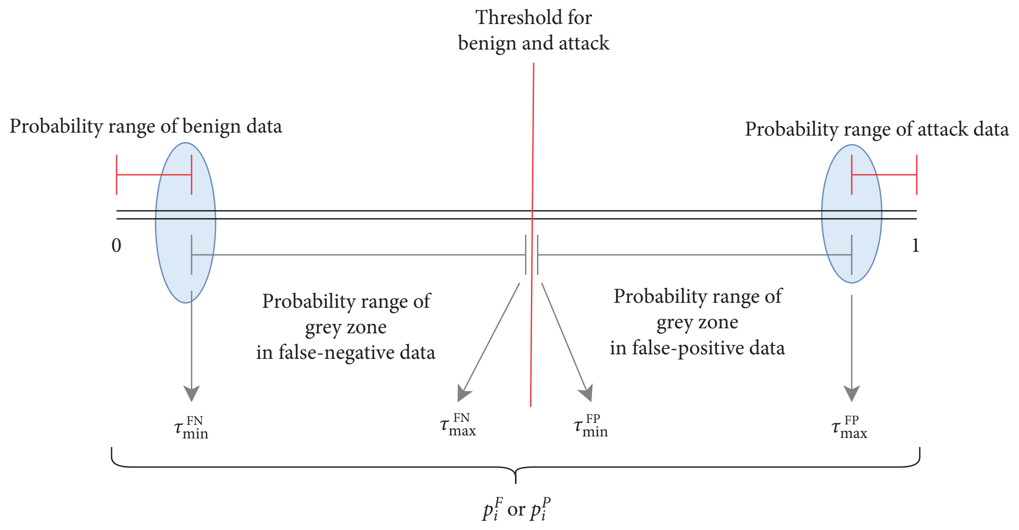

Threshold for benign and attack
Probability range of benign data
Probability range of attack data
0
1
Probability range of grey zone in false-negative data
Probability range of grey zone in false-positive data
$\tau_{\min}^{FN}$
$\tau_{\max}^{FN}$
$\tau_{\min}^{FP}$
$\tau_{\max}^{FP}$
p_i^F or p_i^P

Figure 2: The illustration of the grey zone threshold.

Similarly, to find the threshold for FP data, we apply the technique used for FN data, as shown in Figure 3(b). Using probability percentiles for FP and TP data as inputs, we assign the 0^{th} percentile of FP ($p_{0^{\text{th}}}^{\text{FP}}$) as the minimum value ($p_{\min}^{\text{FP}}$) and the 100th percentile of true positive ($p_{100^{\text{th}}}^{\text{TN}}$) as the maximum value ($p_{\max}^{\text{FP}}$). We then adjust $p_{\max}^{\text{FP}}$ by checking if it is greater than the 0.5th percentile value of true positive $p_{5^{\text{th}}}^{\text{TP}}$. If not, we decrement the percentile by 0.1 and update $p_{\max}^{\text{FP}}$. We continue adjusting until $p_{\max}^{\text{FP}}$ is greater than $p_{0.5^{\text{th}}}^{\text{TP}}$ and accept $p_{\min}^{\text{FP}}$ and $p_{\max}^{\text{FP}}$ as the FP data thresholds, $\tau_{\min}^{\text{FP}}$ and $\tau_{\max}^{\text{FP}}$, respectively.

4.3. Packet Handler.

Figure 4 shows the detailed steps on the packet handler step. We defined packet as S_j and flow as F_i. When the system receives the new packet S_j, it is checked by "belong to existing flow?" step to see whether it belongs to an

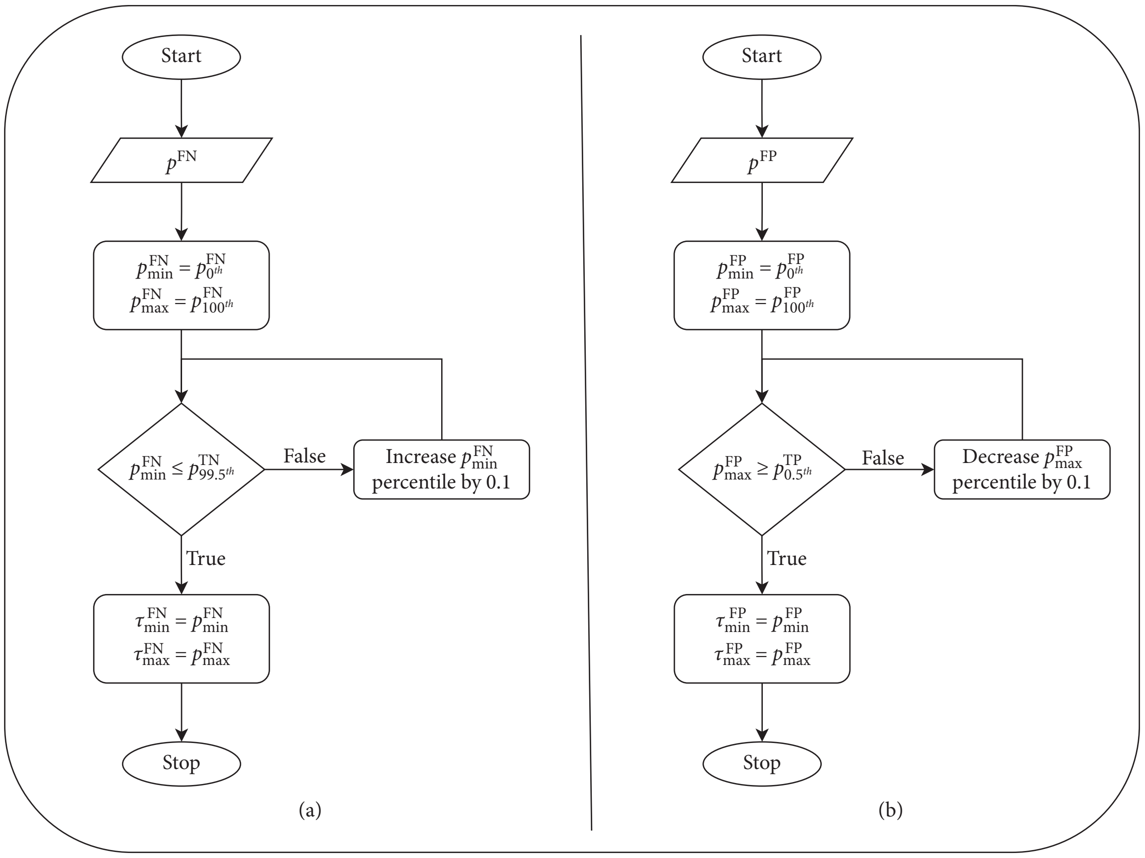

Start
p^{FN}
$p^{FN}_{\min} = p^{FN}_{0^{th}}$
$p^{FN}_{\max} = p^{FN}_{100^{th}}$
$p^{FN}_{\min} \le p^{TN}_{99.5^{th}}$
False
Increase $p^{FN}_{\min}$ percentile by 0.1
True
$\tau^{FN}_{\min} = p^{FN}_{\min}$
$\tau^{FN}_{\max} = p^{FN}_{\max}$
Stop
(a)
Start
p^{FP}
$p^{FP}_{\min} = p^{FP}_{0^{th}}$
$p^{FP}_{\max} = p^{FP}_{100^{th}}$
$p^{FP}_{\max} \ge p^{TP}_{0.5^{th}}$
False
Decrease $p^{FP}_{\max}$ percentile by 0.1
True
$\tau^{FP}_{\min} = p^{FP}_{\min}$
$\tau^{FP}_{\max} = p^{FP}_{\max}$
Stop
(b)

Figure 3: Exhaustive grey zone threshold searching. (a) Threshold in false-negative data. (b) Threshold in false-positive data.

existing flow or not. Furthermore, a flow entry is defined as bidirectional when its address port pair and reverse address port pair belong to the same entry. If the incoming packet S_j has the same pair with the existing flow entry F_i, then it will be grouped in "group S_j to existing flow F_i" step. Otherwise, the system creates the new flow entry F_i for this S_j in the next step and goes back to receive a new packet. After grouping the packet to an existing flow, the subsequent processes occur in parallel. The first parallel process involves the continuous reception of new packets, while the second parallel process gets the total number of flows $|F|$. This total number is used to check how many flows need to be examined by the IDS.

4.4. Flow Checker.

4.4. Flow Checker. Figure 5 shows the detailed steps in the process of deciding whether the data are to be sent to intrusion detection for further processing or not. In the packet handler step, the system has obtained the total number of flows $|F|$. Then, it starts to process the first flow in check the waiting time for a new S_j in F_i step. The waiting time indicates the amount of time that the flow has been waiting for a new packet to arrive since the last packet seen The next step is to determine if the waiting time has reached a defined threshold in waiting time $\ge \tau^{Q}$. If the waiting time has reached the threshold, or if the TCP flag on the last packet indicates either FIN or RST, the flow has been completed. Therefore, the flow features will be extracted in step feature extraction for F_i by aggregating packet information and calculating some statistical data. Finally, in step send F^* to flow-based, the extracted flow data F^* then is sent to flow-based detection for further processing. However, if the waiting period has not yet reached the threshold, it examines the flow duration. If the flow duration reaches the defined threshold, it indicates that the flow is too long, and therefore, the system stops waiting for the next incoming packet for this F_i and directly transfers the data in Send F_i to packet-based step to be processed by packet-based detection. Finally, the system returns to the first looping step to verify the other flows. After examining all flow data, the process ends and returns to the Packet Handler to receive and group the next incoming packet.

4.5. Grey Zone Checking. Figure 6 shows the detailed steps involved in determining whether a flow is in the grey zone data and must be rechecked by packet-based detection or not. After flow-based detection analyzes the flow, it outputs the confidence score p_i^F. The system then takes this score and checks the thresholds in the next steps. $\tau^{FP}_{\min}$, $\tau^{FP}_{\max}$, $\tau^{FN}_{\min}$, and $\tau^{FN}_{\max}$ are the thresholds used to determine the grey zone data. If the score p_i^F is below $\tau^{FP}_{\max}$ or above $\tau^{FP}_{\min}$, or if it falls below $\tau^{FN}_{\max}$ or above $\tau^{FN}_{\min}$, it indicates a low-confidence level in flow-based detection, requiring further verification through packet-based detection However, if the score of p_i^F is not inside the range of those thresholds, it is sent to detection labeling step for classifying the data as benign or anomaly.

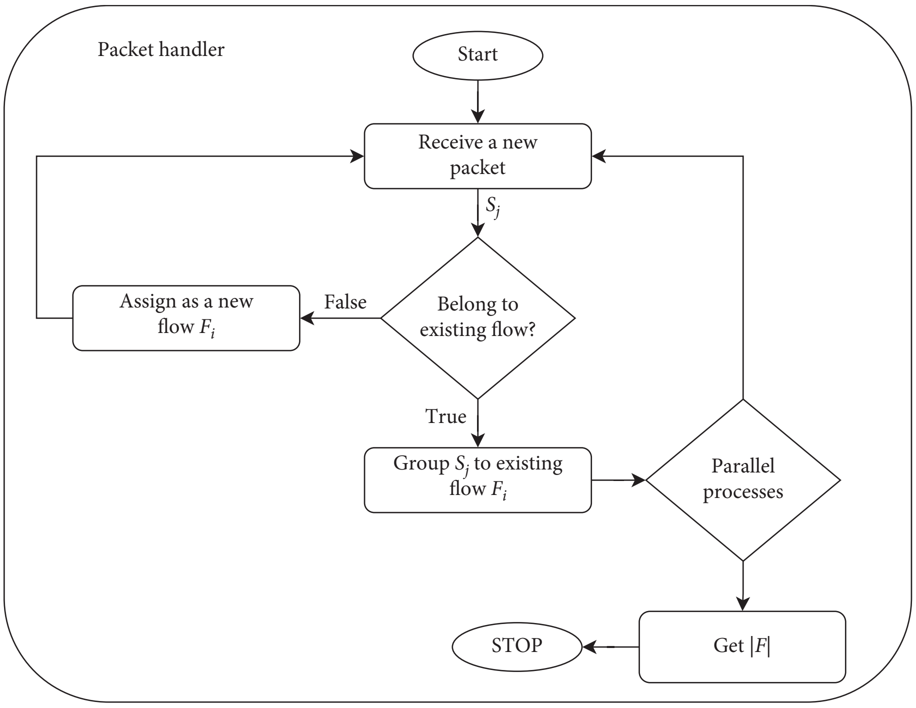


FIGURE 4: The detail of the packet handler.

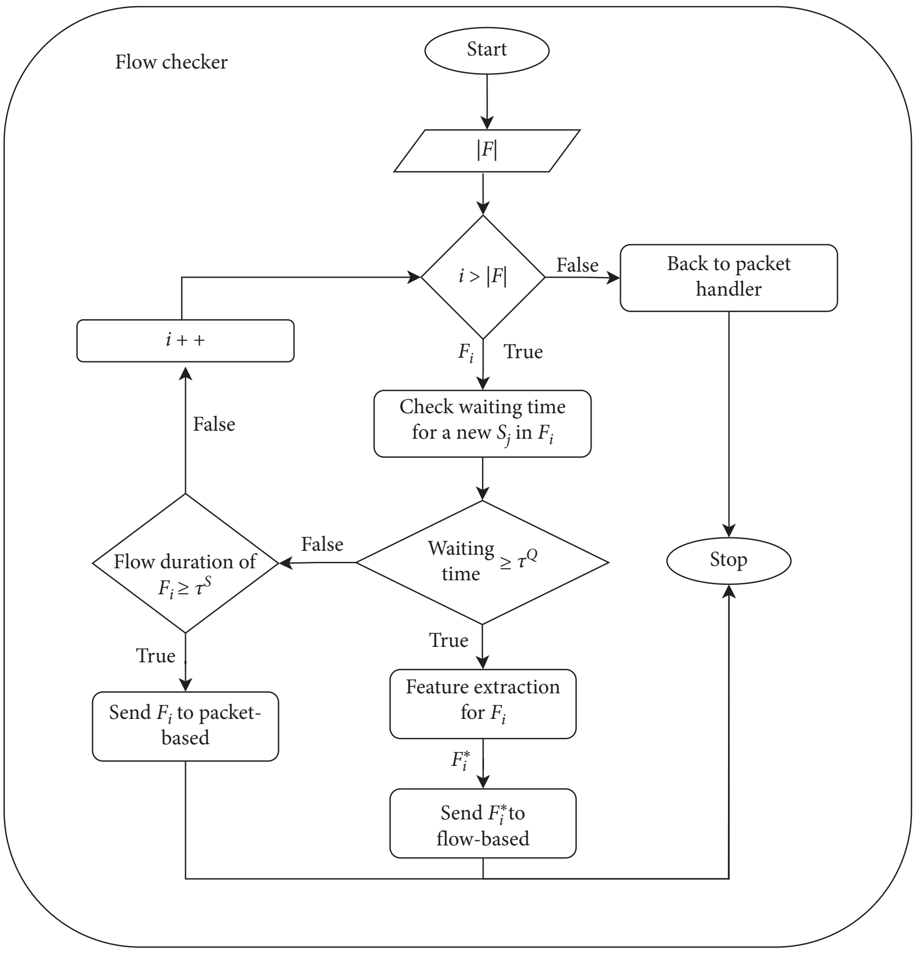

Flow checker
Start
$|F|$
$i > |F|$
False
Back to packet handler
$i++$
F_i
True
Check waiting time for a new S_j in F_i
False
Waiting time $\geq \tau^Q$
False
Flow duration of $F_i \geq \tau^S$
Stop
True
True
Feature extraction for F_i
Send F_i to packet-based
F_i^*
Send F_i^* to flow-based

FIGURE 5: The detail of the flow checker.

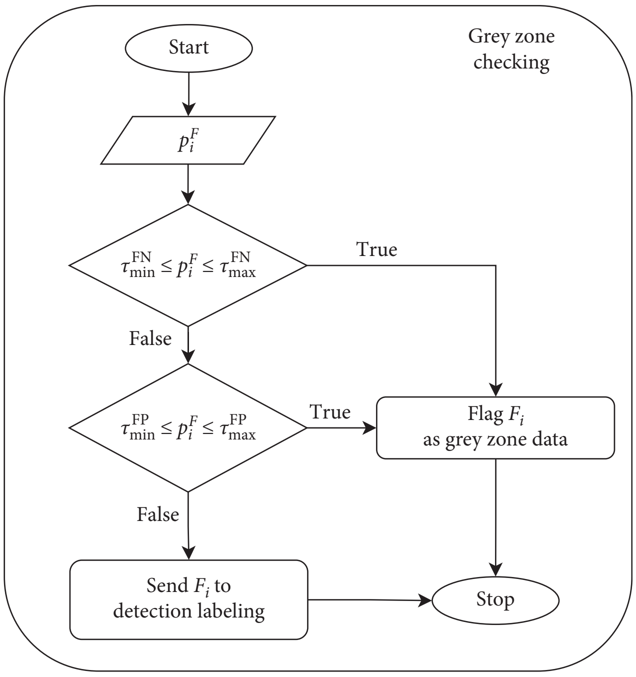

Grey zone checking
Start
p_i^F
$\tau_{\min}^{FN} \le p_i^F \le \tau_{\max}^{FN}$
True
False
$\tau_{\min}^{FP} \le p_i^F \le \tau_{\max}^{FP}$
True
Flag F_i as grey zone data
False
Send F_i to detection labeling
Stop

Figure 6: The detail of grey zone checking.

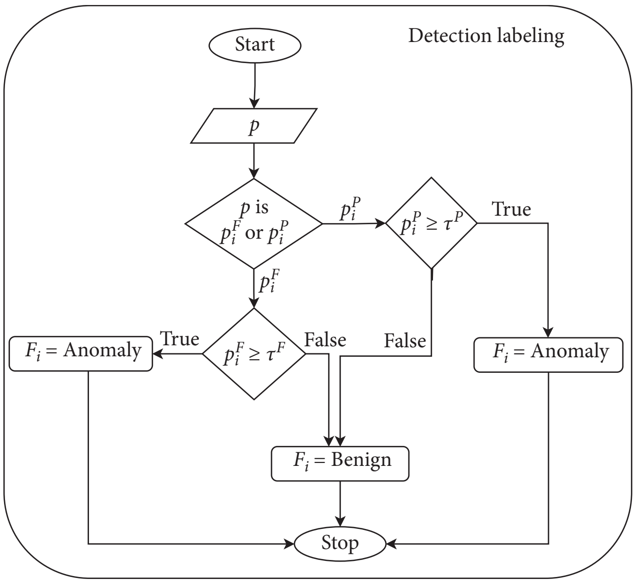


Figure 7: The detail of detection labeling.

4.6. Detection Labeling. Figure 7 shows the detailed steps of detection labeling for classifying the data as benign or anomaly based on the confidence score from either flow-based detection p_i^F or packet-based detection p_i^P. After getting the confidence score, the system checks whether it is p_i^F or p_i^P. If the input is p_i^F, the system checks the score with the defined threshold τ^F. The flow is then classified as malicious if the score of p_i^F is greater than τ^F; otherwise, it is benign. Furthermore, if the input is p_i^P, the system checks the score with the defined threshold τ^P, and if the score is greater than τ^P, then the data are classified as an anomaly.

5. System Implementation

This section provides a detailed explanation of the system implementation. It includes the testbed setup, the method of generating the traffic, the preprocessing method, the DL architecture, and the setup of unknown attacks. All the code to run the proposed solution can be retrieved from our GitHub repository1.

5.1. Testbed Setup. Our testbed environment, shown in Figure 8, used a Kubernetes-based microservices system with 27 services. This setup emulated a microservice system operation and generated both benign and malicious traffic, which was used as the dataset to train our ML models. To generate malicious traffic, we added two pods: an attacker pod initiating malicious activities and a damn vulnerable web application (DVWA) pod with open vulnerabilities. As the existing microservices applications lacked exploitable vulnerabilities, the DVWA was crucial for generating malicious traffic. Additionally, we involved an external attacker in diversifying the attack strategies.

5.2. Traffic Generation. High-quality datasets play a crucial role in developing and evaluating ML models. However, publicly available datasets specific to microservices are limited. To address this gap, we took a systematic approach to create our own dataset.

The dataset collection approach for our study differs significantly from traditional monolithic systems. In monolithic systems, collecting the traffic at the router level often suffices to understand communication patterns since all services are tightly integrated within a single application.

Conversely, a microservice architecture introduces a higher level of complexity. Microservices communicate over networks using diverse protocols, and each microservice operates as a standalone entity with dynamic interactions. To comprehensively understand microservices' intricate communication patterns, we collected traffic at the finer granularity of individual microservices.

By collecting the traffic at each individual microservice, we were allowed to explore the nuances of microservices architecture, revealing variations in protocols, traffic volumes, and unique patterns specific to each service. Such detailed data collection was crucial to construct a dataset that accurately represents the diverse and decentralized nature of microservices interactions.

5.2.1. Benign Traffic. Our first application to simulate benign traffic was a microservices-based e-commerce application by Google [32]. This web application, comprising 11 microservices, lets users explore products, add items to a cart, and make purchases. It also includes a load generator that mimics real shopping activities by continuously sending requests to the front end. The architecture of the e-commerce application is shown in Figure 9.

We further diversified our benign traffic by deploying a second application: a social network application with microservices communicating over Thrift RPCs [33]. Figure 10 shows the architecture of this social network application.

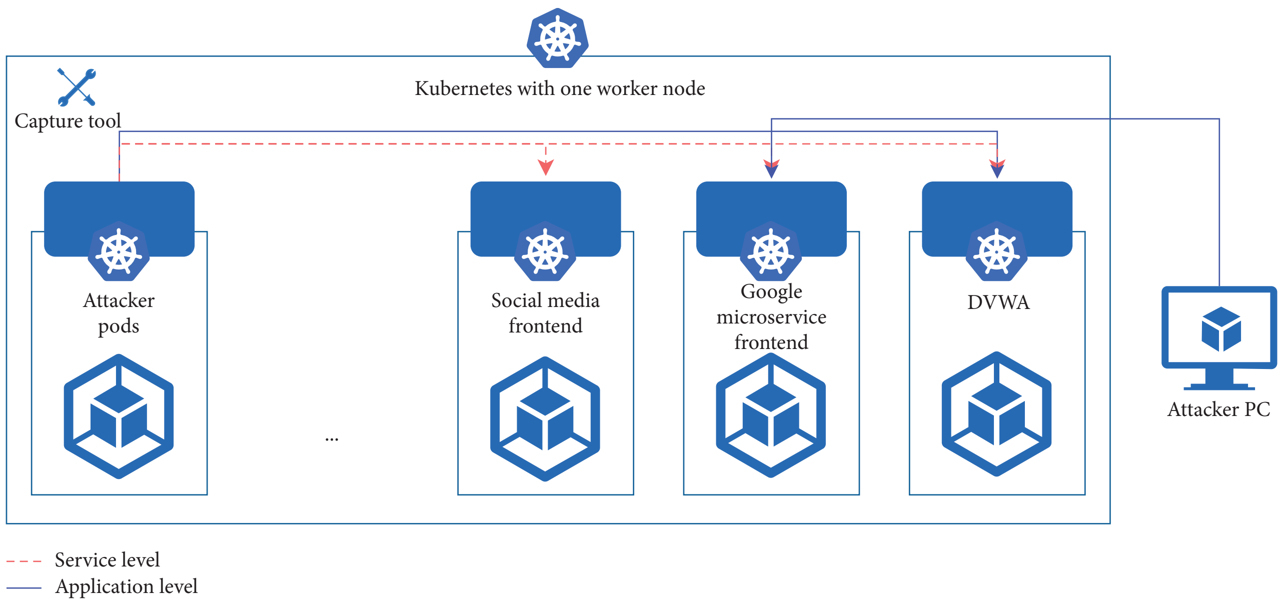


Figure 8: Testbed architecture.

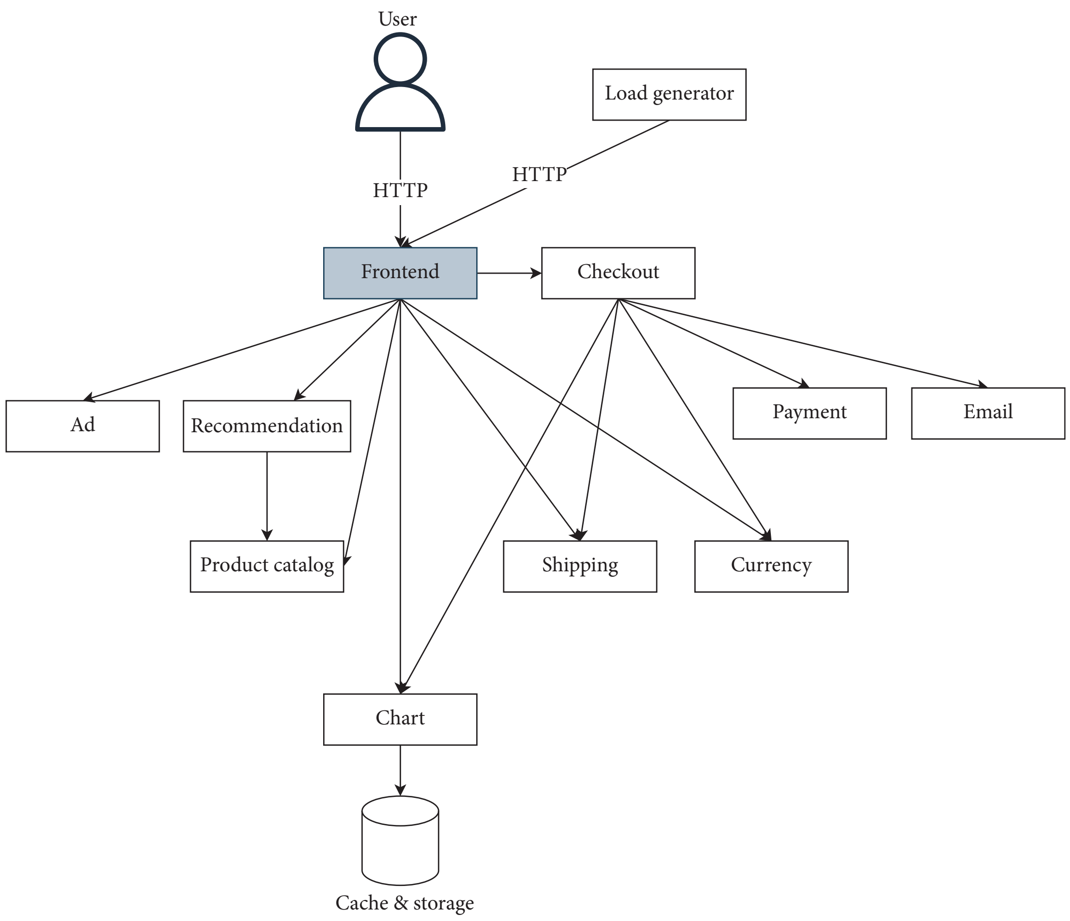


Figure 9: The architecture of e-commerce apps.

5.2.2. Malicious Traffic. This study encompasses various attack scenarios aimed at mimicking malicious traffic and critically assessing microservices security threats. Malicious traffic is primarily classified into two categories: service-level attacks and application-level attacks. The former involves "noisy traffic" attacks that aim to overwhelm the network,

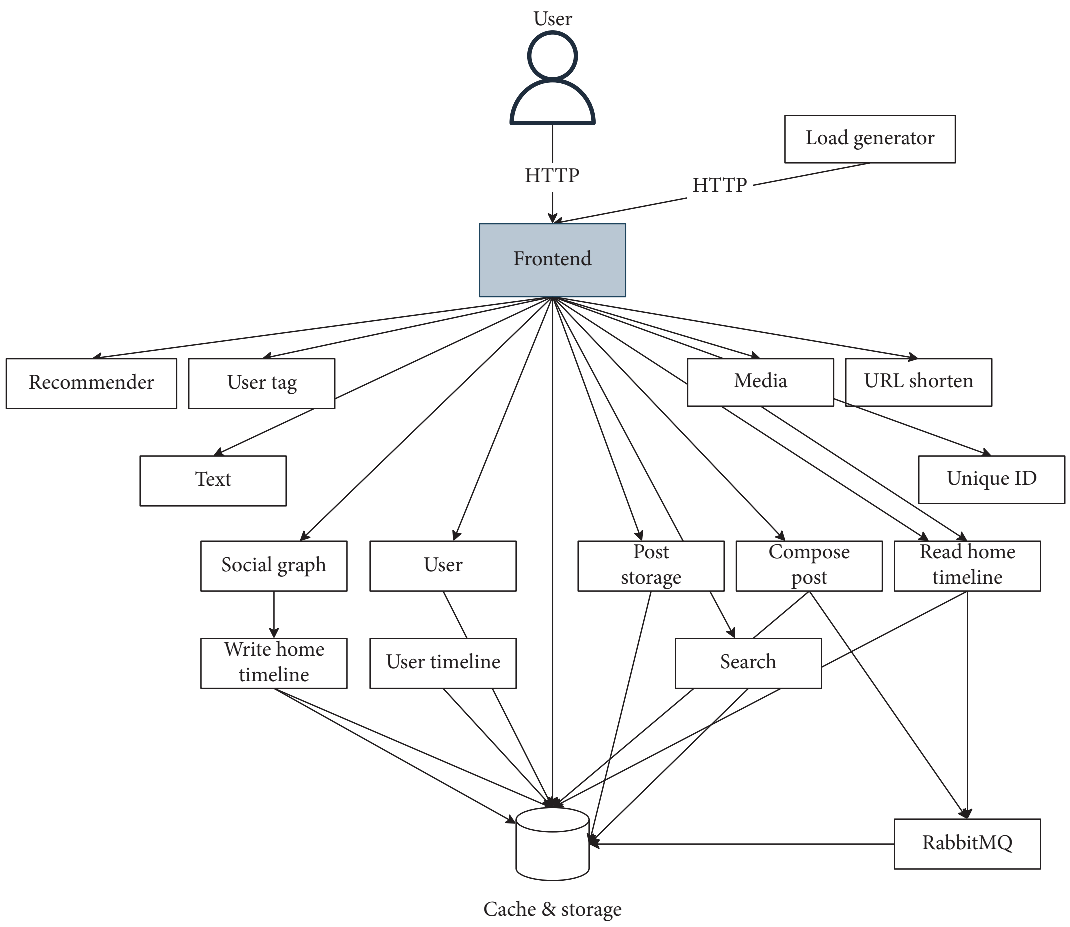


Figure 10: The architecture of social network apps.

while the latter comprises "silent traffic" attacks that subtly blend with normal traffic to exploit application logic.

In simulating service-level malicious traffic, we utilized two attack strategies: brute force and denial-of-service (DoS) attacks. The brute force strategy attempted to overwhelm the target with numerous phrase combinations, employing the DIRB tool to locate hidden server files and directories [34]. In contrast, DoS attacks flooded the target with excessive requests to exhaust its resources, using the HPing tool [35] and Slowloris [36]. The latter exploit concurrent connection limits, disrupting the service availability effectively.

To simulate "silent attacks," we implemented three methods: vulnerability scanning, SQL injection, and cross-site scripting. For vulnerability scanning, we used the Nikto Tool to scan the system thoroughly, identifying active services and possible open ports. We employed SQL injection, a common database security attack, to extract sensitive information by manipulating SQL queries. Lastly, cross-site scripting involved injecting harmful scripts into web applications by sending malicious browser executable codes to the end user.

After completing the traffic generation process, we obtained a total of 126,213 labeled samples, consisting of 73,275 benign and 52,938 malicious instances. The malicious traffic includes 20,507 brute force attack samples, 21,401 scanning attack samples, 5120 SQL injection samples, 5051 XSS samples, and 2859 DoS samples.

5.3. Benchmark Datasets.

5.3. Benchmark Datasets. To ensure a comprehensive and fair performance comparison, we utilized not only our own generated dataset from a microservice environment but also various public datasets. We selected recent public datasets that provide raw PCAP files and detailed documentation of their data collection methods. This documentation includes insights into the tools and techniques used, the data capture environment, and timestamps of attack scenarios, which are essential for our packet-based detection approach that involves raw data processing and re-labeling. Based on these criteria, we identified three datasets that met our requirements. Below is a brief overview of the testing datasets used in this study:

- CIC-IDS-2017: Developed by the Canadian Institute for Cybersecurity in 2017 [37], this dataset features 7 distinct attack classes and includes a total of 2.8 million samples. It was generated over five days using a network of 14 machines and provides raw PCAP files, making it widely used in numerous studies for its comprehensive range of attack scenarios.

- CIC-IDS-2018: Building on its predecessor, the CIC-IDS-2018 dataset was released in 2018 and includes the same attack classes. It expands the testing environment to a cloud setting, involving over 500 machines over 10 days, thereby increasing the complexity and scale of attack simulations. This dataset comprises 9.3 million samples and continues to use the same tools as the CIC-IDS-2017 dataset.
- CREMEv2: Based on the MITRE ATT&CK framework, the CREMEv2 dataset focuses on specific threat models and methodologies. It includes five attack variants, such as ransomware, resource hijacking, mirai, disk wipe, and end point dos. These attacks are developed using 14 tactics across 17 different MITRE techniques, with a total of 2.3 million samples [38].

These datasets serve as a robust testing ground to evaluate the resilience and adaptability of our proposed solution across different environments, ensuring a thorough assessment of its effectiveness.

5.4. Preprocessing.

5.4. Preprocessing. The raw benign and attack traffic data recorded for training must first be preprocessed, involving feature extraction and cleaning. Given the distinct structures of flow-based and packet-based data, we adopted different preprocessing strategies.

5.4.1. Flow-Based Detection. The preprocessing of flow-based data involves three steps. First, we used NFStream [39] to extract 61 features from raw data. After this, we removed features without variance and those related to testbed settings to prevent model bias toward specific patterns. Then, we cleaned the data by removing the missing values and processed the data in the appropriate format. Lastly, we rescaled the data using StandardScaler from Scikit-Learn.

5.4.2. Packet-Based Detection. Figure 11 illustrates the packet-based preprocessing steps, following [40]. The process begins with grouping raw network packets by 5-tuple for flow identification. Trace sanitization then removes MAC and IP addresses of each packet, preventing the model from learning specific testbed configurations. Afterward, flow cleaning is performed to remove identical or duplicated data, which could cause bias in DL model training.

The final step in our process is uniform-length trimming. We extract and trim the first n-packets from each flow. This not only lessens the traffic load for analysis but also accelerates the process for long sessions. As DL models require consistent data lengths, we set a default packet size. Packets exceeding this setup are trimmed, while smaller ones are zero-padded. Previous research suggested that the first three packets, each being 60 bytes long, yielded optimal results [40]. These trimmed bytes are considered as one-dimensional vectors and used as CNN model inputs.

5.5. Key Features Tool. In this work, we delve into an in-depth investigation of the key features extracted from our CNN model to elucidate the mechanisms by which our CNN model determines the detection result. This is crucial not only for model validation but also to enhance transparency and interpretability, which are often obscured in DL models. For this reason, we employed SHapley Additive exPlanations (SHAP) to extract and elucidate these key features.

SHAP is a game-theoretic approach to explain the output of any ML model. It determines each feature's importance by approximating the contribution of each feature to every possible prediction. In the context of CNN, SHAP meticulously identifies and ranks key features by assessing the impact of each convolutional layer and its neurons on the final prediction. This methodology is preferred because it offers insightful, consistent, and locally accurate attributions, making it superior in capturing intricate data patterns and dependencies.

5.6. Unknown Attack. In the context of our study, the term "unknown attack" refers to attack scenarios that are novel to our detection system. These scenarios simulate real-world conditions where defense mechanisms encounter unknown threats.

Furthermore, research has shown that ML-based IDS are vulnerable to adversarial perturbations. While adversarial attacks could theoretically reshape packet-level features, doing so without disrupting the attacks effectiveness is highly impractical, especially in black-box scenarios where the attacker lacks access to the IDS feature extraction process. Most adversarial research assumes attacks occur at the feature level, requiring privileged access to IDS devices or network infrastructure, making real-world execution unlikely unless the attacker controls the IDS itself.

Instead, real-world adversarial tactics are more likely to involve host-based perturbations, where attackers modify their behavior at the command level rather than manipulating network features postcapture. These behavioral modifications can alter network activity in ways that cause an ML-based IDS to fail in recognizing the attack, as the observed patterns deviate from the training data.

To account for these adversarial behaviors, our study evaluates the models robustness by introducing three categories of unknown attacks: (1) variant attacks, which resemble known threats but use different execution techniques to evade detection; (2) anomalies that mimic benign traffic, designed to blend in and bypass traditional detection mechanisms; and (3) completely novel attack vectors, representing threats the model has never encountered. By testing across these categories, we ensure that our model can effectively detect both adversarial and unknown attack behaviors, reinforcing its resilience and generalization capabilities in real-world scenarios.

We focused on SQL injections as the first type of unknown attack. We trained the CNN model using standard SQL injections, where malicious SQL statements are directly

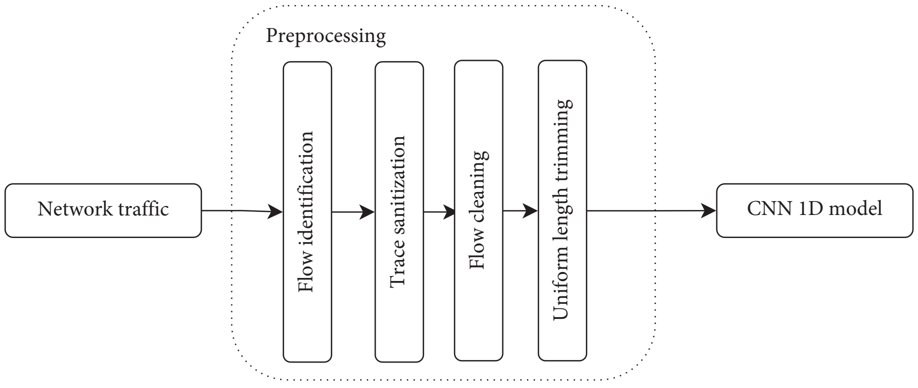


Figure 11: Packet-based preprocessing steps.

inserted into entry fields, causing the database to execute them. These often trigger error messages that can reveal insights about the database structure. We then tested with blind SQL injections. Unlike standard SQL injections, blind SQL injections do not produce direct database error responses but rather require interpretation of subtle changes in application responses. Despite similar objectives to exploit system vulnerabilities, their differing methodologies make blind SQL injections a novel unknown attack exploit, presenting a unique challenge to traditional detection methods.

The second type of unknown attack simulates benign behavior, initially involving normal login form access. The anomaly arises when the attacker launches a brute force attack on the login form within the social media application.

The third type of unknown attack targets microservices applications using several methods. These attacks include Cross-Site Request Forgery (CSRF), File Upload, File Inclusion, and Command Injection attacks. In CSRF, a malicious actor manipulates a user's browser to execute unauthorized actions on a website where the user is authenticated. File Upload attack exploits an application's vulnerabilities, enabling the upload and execution of harmful files on the target system. File Inclusion attack gains unauthorized access by exploiting vulnerabilities to execute server files. Lastly, a Command Injection attack allows arbitrary command execution and potential unauthorized system control by inserting malicious commands into a vulnerable application. All these methods aim to exploit system vulnerabilities, breach security, and gain control.

6. Results and Analysis

This section presents the numerical results obtained from the tests that were carried out in order to evaluate the performance and effectiveness of our DL models.

6.1. Baseline Configurations DL Architectures. In our research, we employed CNN due to its excellent autoprofiling abilities. CNNs are especially appropriate due to their proficiency in handling large and complex datasets, where recognizing patterns and anomalies is crucial for effective intrusion detection. The architecture of CNNs supports the efficient processing of both statistical features derived from flow-based data and the sequential data characteristic of packet-based detection. This dual capability aligns perfectly with our unified approach, which seeks to enhance both accuracy and computational efficiency by combining flow- and packet-based methods.

Furthermore, CNNs have been widely adopted in intrusion detection research, as comprehensively surveyed in [41]. These prior studies have consistently shown that CNNs can effectively detect and classify anomalies and malicious activities, providing a strong empirical foundation for our decision to employ this technology.

Table 3 shows the architecture of our CNN model, which consists of three convolution layers for critical feature extraction and learning. The model also includes a pooling layer, a dropout layer, and a flattened layer, followed by three dense layers for classification, with a sigmoid function as the final activation step. To optimize the model performance, we employed Optuna, a Bayesian optimization framework, for hyperparameter tuning [42]. The tuning process explored a search space for key parameters, including the learning rate, batch size, kernel size, number of convolutional filters, dropout rate, and weight initialization strategies. Optuna's Tree-structured Parzen Estimator (TPE) algorithm was used to efficiently navigate the hyperparameter space, selecting configurations that maximized detection accuracy while minimizing overfitting. This automated tuning process ensured that our CNN architecture was optimally configured for intrusion detection.

6.1.1. Dataset Configuration. As previously mentioned, dataset creation involves generating two types of traffic: benign and malicious. This ensures a comprehensive dataset, covering various scenarios and capturing microservices application nuances in normal and malicious behaviors.

The dataset was split into three parts: 70% for the training set, 15% for the testing set, and 15% for the grey zone investigation set. The grey zone investigation set was used for a more comprehensive exploration of output probabilities, thereby increasing the diversity and range of these probabilities. By allocating a dedicated grey zone investigation set, we could effectively evaluate and fine-tune the system's performance within this critical zone, ensuring robust and accurate results.

Table 3: Deep learning architectures.

Layer	Type	Filters/neurons/stride	Padding
1	1D-ConV + ReLu	32 (kernel size = 3)	Same
2	1D-ConV + ReLu	64 (kernel size = 3)	Same
3	1D-ConV + ReLu	128 (kernel size = 3)	Same
4	MaxPooling + dropout + flatten	Kernel size = 3, dropout = 0.5, stride = 2	Same
5	Dense + ReLu	256	—
6	Dense + ReLu	512	—
7	Dense + sigmoid	1	—

6.1.2. Evaluation Strategy. The evaluation of the models in this study focuses on two main aspects: their classification performance and computational complexity. The latter is measured using the Linux-based PERF tool, which calculates the number of instructions required to process a single flow, while for the former, we used $F1$ score, precision, recall, and accuracy.

6.2. Detection Performance Flow-Based Detection. A DL model necessitates a threshold for data classification. To achieve an optimal balance between precision and recall, and thus maximize the $F1$ score, we employed an exhaustive search strategy. This strategy enabled us to identify the most suitable threshold of detection for our model, which was found to be 0.513222. Hence, any score below this threshold is classified as benign, while scores above this threshold are marked as malicious.

Figure 12 displays the confusion matrix of our model's flow-based performance. The model achieved a high accuracy of 98.2% in correctly identifying benign traffic but resulted in a 1.8% FN rate, indicating 122 instances of incorrectly classified data. For attack traffic, the model's accuracy was even higher at 99.2%, with a 0.8% FP rate. Analyzing the FPs revealed that most of them were caused by noisy attacks, specifically DoS attacks.

We conducted two types of DoS attacks: ICMP and HTTP. The flow-based detection method struggled to accurately classify HTTP-based attacks due to the similarity in statistical information between malicious and normal requests.

6.2.1. Packet-Based Detection. Figure 13 illustrates the confusion matrix for the packet-based detection method. It clearly exhibits superior performance compared to the flow-based method. The packet-based approach achieves an impressively low FN rate of 0.1%, and an accuracy rate of 100% in classifying malicious traffic, indicating its high precision in correctly identifying such traffic. Moreover, this method proves that examining just three packets within a flow is sufficient to differentiate between traffic types. This greatly reduces the amount of data required for processing, leading to more efficient analysis.

The packet-based method also outperforms the statistical features used in the flow-based approach by leveraging automatic feature learning. By adaptively and automatically learning spatial hierarchies of features through the convolutional layer, this approach can detect a vast array of

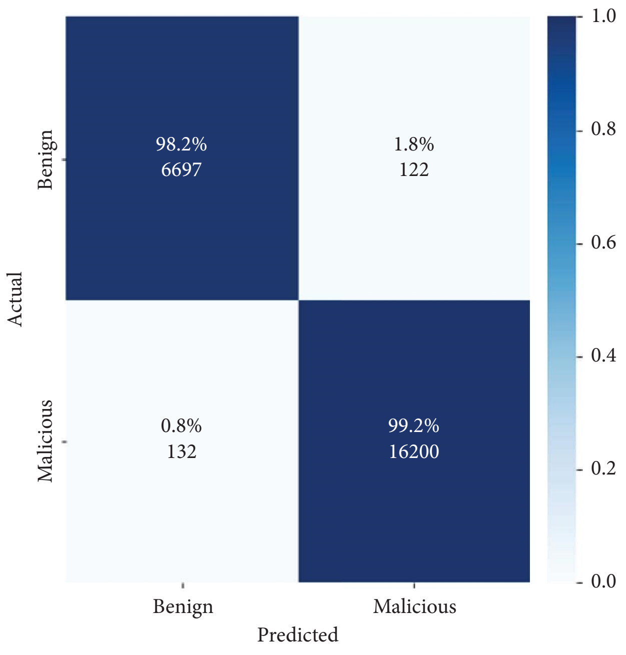


Figure 12: Flow-based performance.

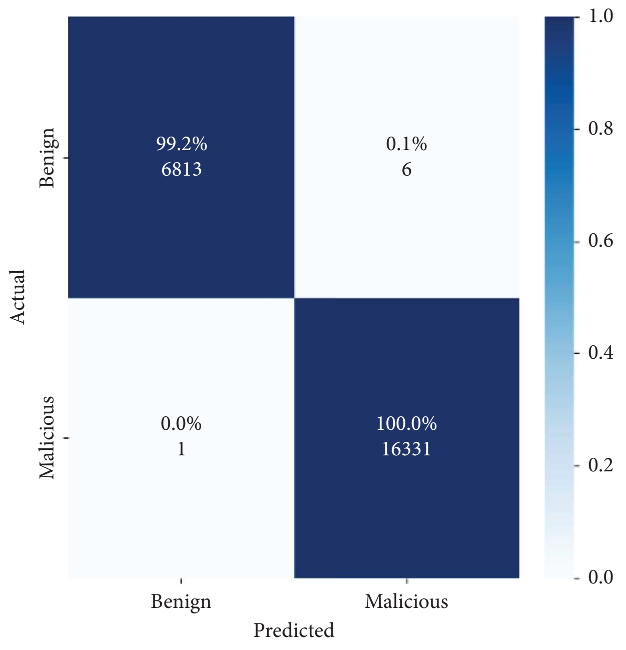


Figure 13: Packet-based performance.

features that may occur anywhere in the traffic, further boosting its effectiveness.

6.3. Key Features of the CNN Model

6.3. Key Features of the CNN Model. In a CNN model, key features refer to the specific patterns or characteristics in the input data that the model deems most important for making predictions. A CNN model is designed to automatically learn the key features from the input data during the training process. Through a series of convolutional and pooling layers, the model learns to extract and recognize hierarchical features at different levels of abstraction. We then extracted the key features by utilizing SHAP.

In this subsection, we explain the key features used by the CNN model in flow-based and packet-based detection.

6.3.1. Key Features in Flow-Based Detection. The key features extracted by the CNN model using NFStream represent the most relevant statistical characteristics of the flow-based data that are crucial for distinguishing between benign and malicious traffic. These features are essentially the unique patterns and properties that the model has learned to associate with each class, allowing it to accurately classify new traffic samples into the appropriate category. Figures 14(a) and b show the key features of flow-based for benign and malicious traffic. This visualization offers insights into how specific features influence the model's predictions and contribute to its accuracy.

The y-axis of the figure displays a list of features, with more important features situated at the top and less important ones toward the bottom. The x-axis represents the impact of the features on the model's predictions. Features on the right side contribute positively to increasing predictions, while those on the left contribute negatively.

The color coding within the bars adds an extra layer of information. Darker shades of blue correspond to lower feature values, while darker shades of red indicate higher feature values. This color scheme helps visualize the relationship between feature values and their impact on predictions.

In our analysis of these key features, we observed that those key features are primarily associated with maximum packet size within a flow, standard deviation of packet size within a flow, minimum packet size within a flow, and the mean size of a flow. The following offers an in-depth examination of these essential characteristics.

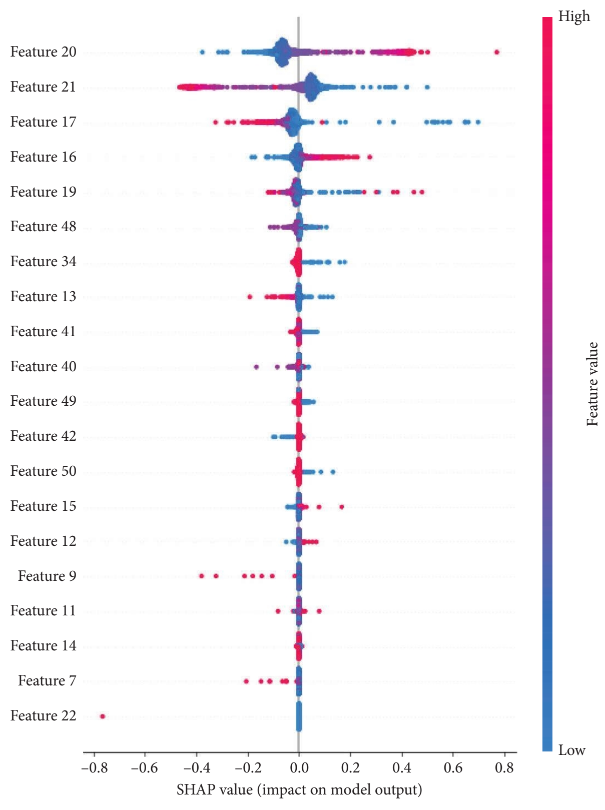
- Maximum packet size within a flow: This feature is indicative of the largest singular data payload exchanged during a communication session. Malicious activities, especially those trying to exploit vulnerabilities, exfiltrate data, or upload malicious payloads, can often lead to packets that have atypical sizes compared to regular traffic. Benign traffic in our microservices environment, in general, tends to have maximum packet sizes that conform to expected norms.
- Standard deviation of packet size within a flow: Variability in the sizes of packets within a flow can be a sign of irregular or anomalous behavior. For instance, a DDoS attack using varying packet sizes or malware trying to hide its activities among regular traffic could lead to a higher standard deviation. Conversely, benign traffic, especially that of well-defined microservices, might exhibit a more consistent and predictable pattern, resulting in a lower standard deviation.
- Minimum packet size within a flow: Some malicious activities involve sending a flurry of smaller, often identical, packets to overwhelm a system (e.g., a type of DoS attack). This can result in a lowered minimum packet size for the flow. On the other hand, benign flows may occasionally have smaller control or acknowledgment packets, but malicious activities might be more pronounced in this aspect.
- Mean size of a flow: The average size of packets within a flow can provide insights into the overall nature of the communication. A flow with unusually large mean packet sizes might indicate data exfiltration or bulk transfer of malicious payloads, while a very low mean might suggest flooding attacks. Benign communications, depending on the application, tend to have mean packet sizes that are reflective of regular data exchanges, neither too small nor excessively large.

In conclusion, these flow-based features capture crucial aspects of the data communication patterns within a network. The subtle nuances and variations in these features between benign and malicious flows provide valuable insights, enabling a model to effectively discern between regular and potentially malicious traffic.

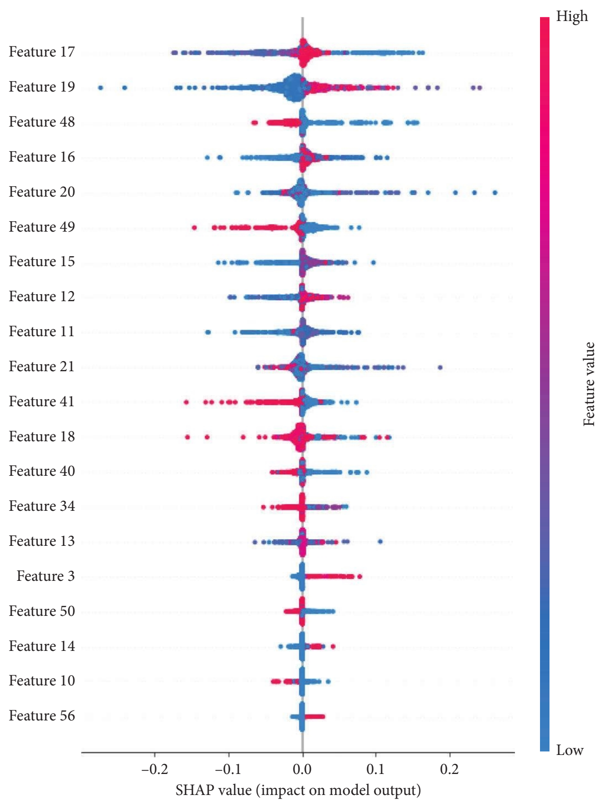
Furthermore, we carried out an exploration of the data distribution of these four key features, as shown in Figure 15. It is evident from the results that the distribution of data differs significantly between benign and malicious data. This observation supports our earlier explanations, emphasizing that benign traffic in a microservice environment demonstrates a more consistent and predictable pattern, while malicious traffic has a more diverse pattern. The clear separation observed in the data distribution is attributed to the model's ability to effectively differentiate between the distinct data patterns associated with benign and malicious activities.

6.3.2. Key Features in Packet-Based Detection. In this section, we explore the key features in packet-based detection, which involves a dataset of 180 features extracted from 3 packets with 60 bytes each in every flow. Our analysis focuses on identifying the key features that the CNN model utilizes to determine the classification outcome. Figure 16 illustrates the significant features used by the CNN in distinguishing between benign and malicious data within packet-based detection.

Upon analyzing the results, it becomes evident that several features significantly impact the CNN's classification ability for benign and malicious traffic. In our deep analysis of these key features, we found that those key features are



(a)



(b)

FIGURE 14: Key features in flow-based detection. (a) Benign traffic. (b) Malicious traffic.

predominantly linked to the destination port within a flow, TCP window size value, and TCP sequence numbers.

Destination port within a flow plays a pivotal role in distinguishing between benign and malicious traffic. Specifically, the CNN model has learned that certain combinations destination ports are indicative of benign or malicious activities, leading to higher predictions when such patterns are detected.

Furthermore, the TCP window size value represents the amount of data (in bytes) that the receiver is willing to accept and buffer at any given time. This feature affects the flow of data between devices. The model identifies specific window size values that correlate with malicious behavior, contributing significantly to the model's predictive accuracy.

TCP sequence numbers also contribute notably to the model's predictions. The sequence numbers of TCP packets hold valuable information about the order and reliability of data transmission. While TCP's initial sequence number (ISN) is randomly generated for security reasons, subsequent sequence numbers during a TCP connection increment predictably for each transmitted packet. This progression can be valuable information for a CNN, which might detect patterns in these sequence numbers that differ between benign and malicious traffic. Furthermore, DL models, especially CNNs, excel at identifying complex interactions between features. The sequence number might not be important on its own but in combination with other packet attributes. For example, analyzing the relative differences between sequence numbers in a data stream, combined with other features, may provide valuable insights.

The interpretability provided by the SHAP summary plot enhances our understanding of the CNN's inner workings. It sheds light on the specific features that contribute to its high classification accuracy in identifying malicious network activity.

Moreover, we also conducted an investigation into the distribution of key data features and specifically focused on some key features, such as features 94, 96, 109, 157, 169, and 179. Figure 17 presents the data distribution between those features. From the figure, it can be seen that those features exhibit more varied distributions. Features with varied distributions can give the model more information to distinguish between different classes. The variation allows the model to capture unique patterns associated with each class, leading to improve the accuracy of the model.

6.4. Unified ML Approach: Flow and Packet. We have organized the explanations in this section into two subsections. The first subsection presents an analysis of the results obtained from finding the grey zone threshold. This threshold is used to filter misclassified data from the flow-based detection and then re-evaluate it in the packet-based detection. In the second subsection, we provide a detailed explanation of the results achieved when employing the unified flow-and-packet technique.

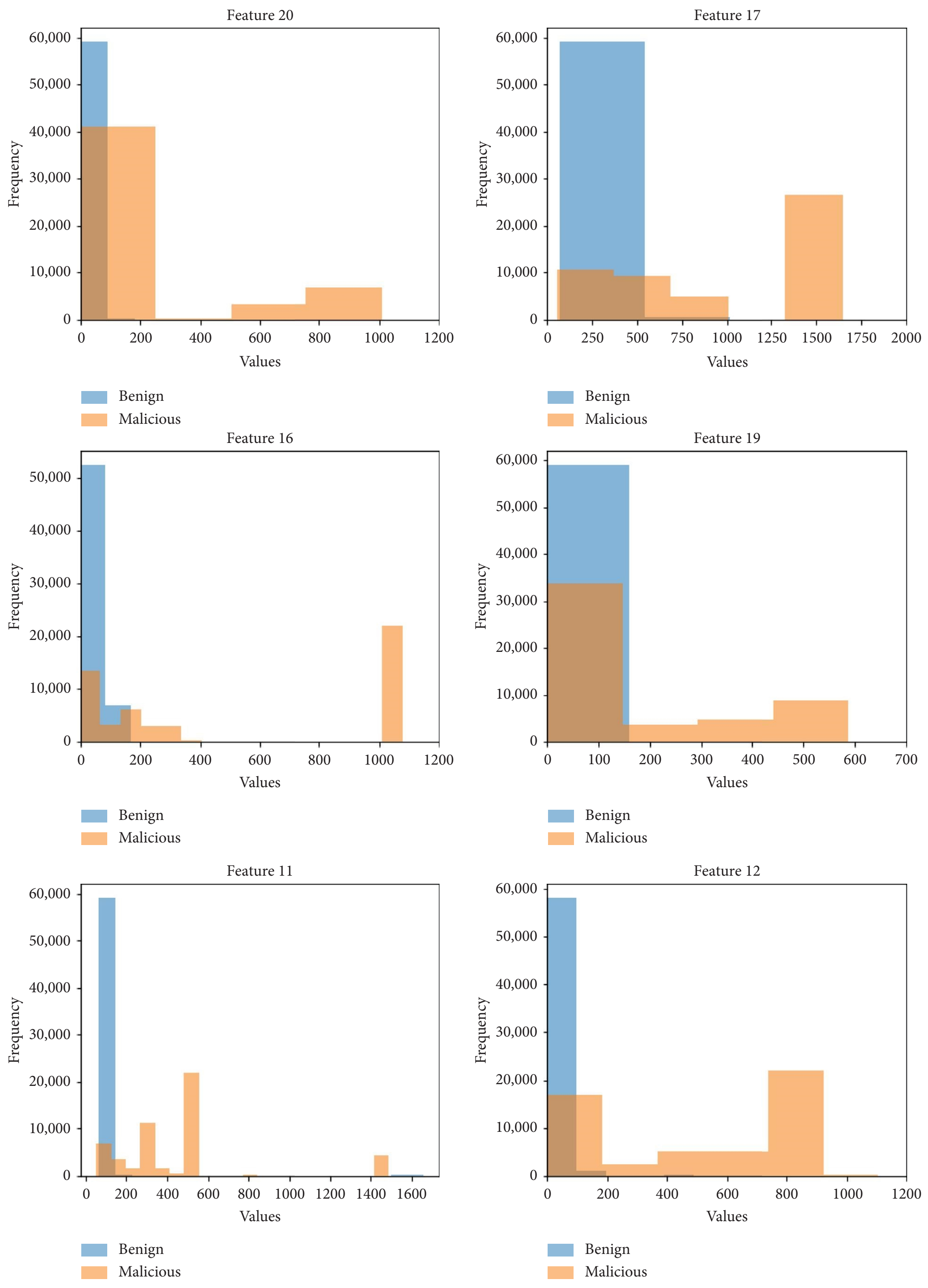
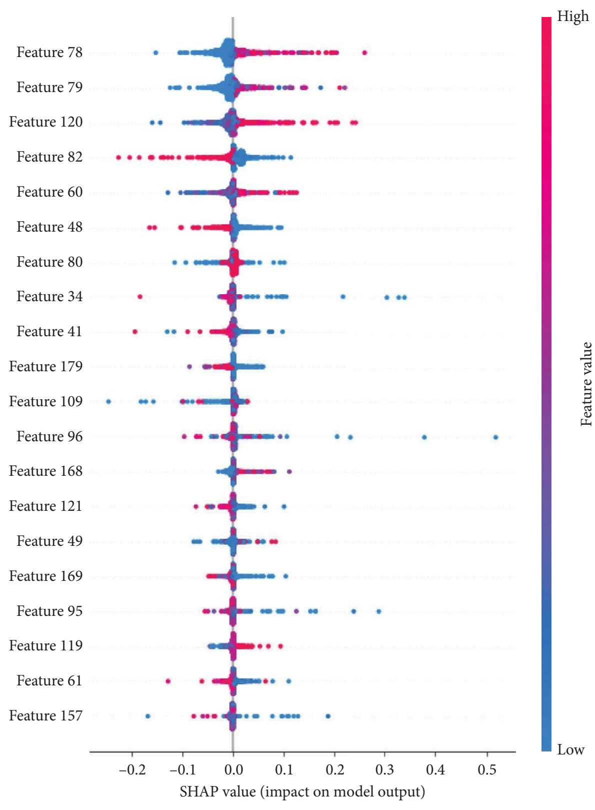
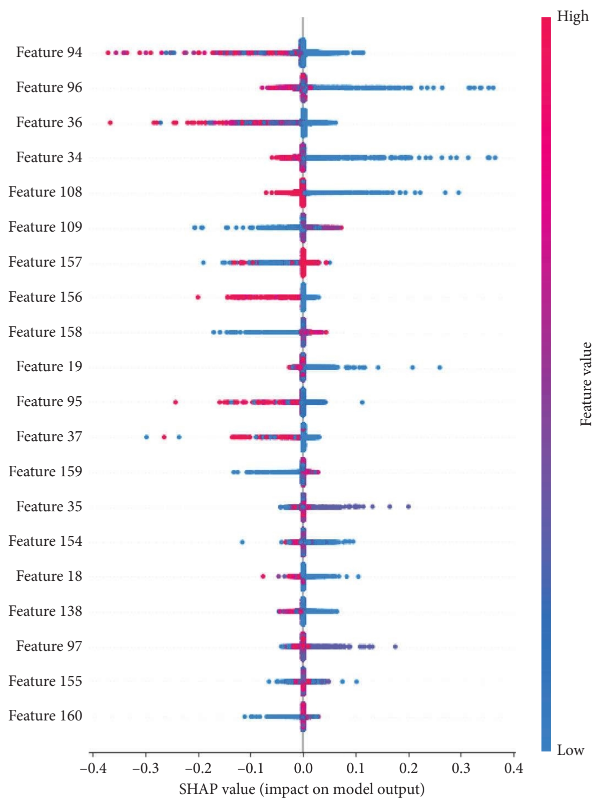


Figure 15: Distribution of features in flow-based data.

6.4.1. Grey Zone Threshold. To address the potential risk of overfitting while tuning the grey zone thresholds, we adopted a methodical approach leveraging separate datasets. Initially, we trained our model using a designated training dataset, ensuring that the model learned the underlying patterns without any influence from the subsequent datasets. Following the training phase, instead of directly employing the trained model on our testing data, we introduced an intermediary step. We used a distinct grey zone dataset to determine the optimal thresholds for classifying uncertain data points. This approach ensures that our thresholds are not biased by the characteristics of the training data.



(a)



(b)

Figure 16: Key features in packet-based. (a) Benign traffic. (b) Malicious traffic.

Furthermore, using a separate dataset for threshold determination acts as an additional validation layer, allowing us to gauge the threshold's effectiveness on unseen data before actual performance testing. Finally, with our model trained and thresholds determined, we evaluated the entire system's performance using a separate testing dataset. This segregation ensures that at each step training, threshold tuning, and testing the data involved is independent, minimizing the risk of overfitting and providing a more holistic and unbiased view of the system's performance.

The grey zone dataset consists of 74,532 benign flows and 64,588 malicious flows, which were analyzed using the flow-based method. The flow-based method provided detection probabilities for each flow, and upon evaluation, we identified 501 instances of FNs and 1398 instances of FPs in the dataset.

To find the optimal grey zone threshold, we conducted an exhaustive search algorithm as shown in Figure 3. The results showed that the default threshold at the 0th percentile successfully captured all FN data, as shown in Table 4. Although 112 true-negative data points were also flagged for re-checking, this trade-off was deemed acceptable as they constituted only 0.5% of total true-negative data.

For the threshold values of τ^{FP}, the default threshold at the 99.99th percentile effectively captured almost all instances of FP data, as shown in Table 5. However, this resulted in 375 true-positive data points being flagged for re-checking, which remained below the 0.5% threshold of the total true-positive data. During the testing phase of our unified flow-and-packet ML model, we then used the defined thresholds.

6.4.2. *Flow vs. Packet vs. Unified.*

Our unified approach employs a two-step approach for detection. Initially, flow-based analysis is conducted, resulting in 122 FNs and 132 FPs, as depicted on the left side of Figure 18. To enhance the accuracy, we use the grey zone threshold to filter and forward 452 data points to the subsequent packet-based analysis for re-evaluation. The right side of Figure 18 shows the updated confusion matrix after re-correction by the packet-based method, revealing significant improvement with no FPs and only one FN remaining. This unified approach effectively enhances the overall performance of our detection model.

Table 6 presents the detection performance of our models: flow, packet, and unified. The unified approach stands out with the best performance, achieving an $F1$ score and recall of 100%. It successfully rectifies incorrect classifications from the initial stage, leveraging the strengths of both flow and packet–based detection. The unified approach's exceptional performance highlights the advantages of this integrated approach.

Notably, the unified approach outperforms the other two, achieving a perfect score of 100% in both $F1$ score and recall. This showcases its ability to correct misclassifications from the initial detection stage, validating its effectiveness.

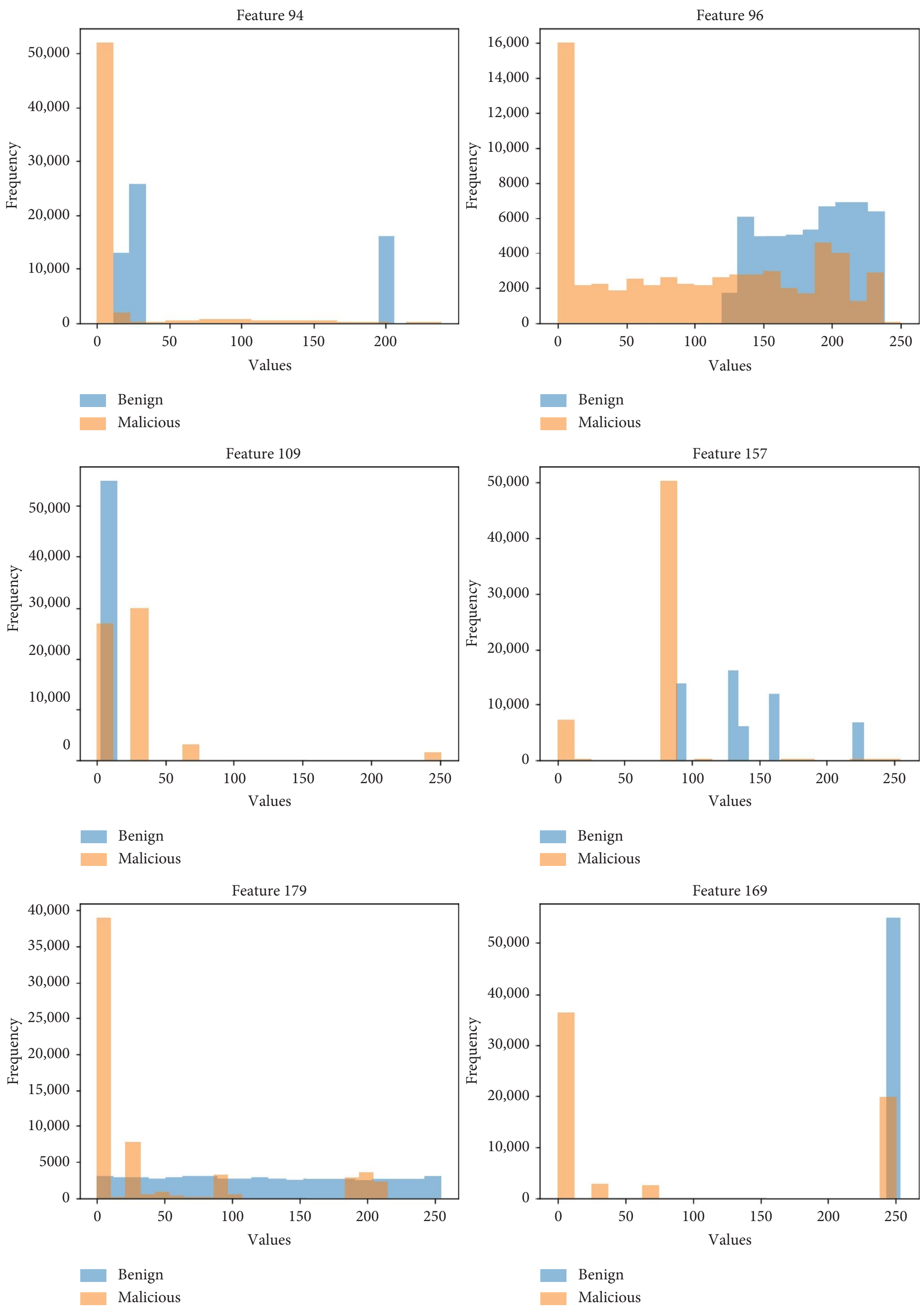


Figure 17: Distribution of features in packet-based data.

The unified approach effectively combines the strengths of flow and packet–based approaches, resulting in superior performance.

Table 7 presents a comparative analysis of our method, MEMBER [8], and CNN–LSTM [25], which are among the most closely related works to ours. The results clearly demonstrate that our approach consistently outperforms both MEMBER and CNN–LSTM across multiple datasets. For example, in our self-generated dataset from a microservice environment, our method achieves an $F1$ score, precision, and recall of 100%, significantly exceeding MEMBER's scores of 99.00%, 98.5%, and 99.5%, and

Table 4: $\tau_{\min}^{\mathrm{FN}}$ threshold values.

Percentile	$\tau_{\min}^{\mathrm{FN}}$ value	True-negative data	False-negative data
0	1.97E – 05	112	501
0.1st	0.00541815	8	500
0.5th	0.0108242	8	498
1st	0.0166902	6	496
2nd	0.0167551	6	492
3rd	0.157455	0	486

Table 5: $\tau_{\max}^{\mathrm{FP}}$ threshold values.

Percentile	$\tau_{\max}^{\mathrm{FP}}$ value	True-positive data	False-positive data
99.99th	0.999982	375	1397
80th	0.999982	375	1397
17th	0.999919	274	244
15th	0.619065	74	210
13th	0.578193	44	193
12th	0.578167	0	189

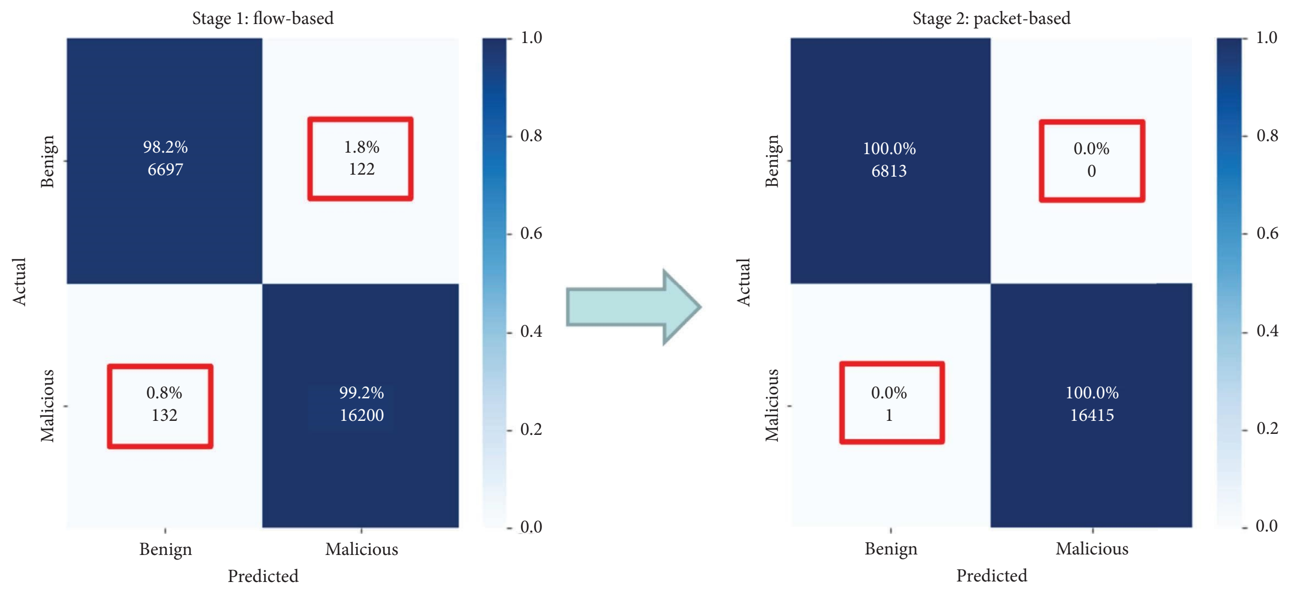


Figure 18: Unified approach detection results.

Table 6: Flow vs. packet vs. unified.

Data source	*F*1 score (%)	Precision (%)	Recall (%)	Number of instructions/flow
Flow	99.22	99.19	99.25	15,635,967
Packet	99.98	99.99	99.96	672,117,601
Unified	100	99.99	100	27,801,023

CNN–LSTM’s scores of 98.8%, 99.87%, and 97.75%, respectively. Similarly, this superior performance is also evident in the CIC-IDS-2017, CIC-IDS-2018, and CREMEv2 datasets.

Our unified model is geared toward adaptability and scalability, which are critical for the dynamic environments in which IDSs are deployed. In such environments, the ability to scale with lower resource consumption is often more valuable than small increments in accuracy. Our unified approach proves to be more efficient, achieving comparable performance while requiring fewer computational resources, as shown in Figure 19. Our system efficiently utilizes the flow-based model as the primary defense most of the time, which demands fewer computational resources, and employs the packet-based model, which requires more computational resources, as a secondary defense

Table 7: Overall performance comparison (in percentages) between our work and the closest-related work across several datasets.

Dataset	Method	*F*1 score (%)	Precision (%)	Recall (%)
Microservices (self-generated)	MEMBER [8]	99	98.5	99.5
	CNN–LSTM [25]	98.8	99.87	97.75
	Ours	**100**	**99.99**	**100**
CIC-IDS-2017	MEMBER [8]	99.06	98.42	**99.7**
	CNN–LSTM [25]	99.25	97.00	98.1
	Ours	**99.68**	**99.83**	99.54
CIC-IDS-2018	MEMBER [8]	98.78	98.22	99.34
	CNN–LSTM [25]	99.32	98.52	98.92
	Ours	**100**	**100**	**100**
CREMEv2	MEMBER [8]	97.67	96.71	98.65
	CNN–LSTM [25]	98.55	97.02	97.78
	Ours	**99.61**	**97.69**	**99.83**

Note: Bold values indicate the best result.

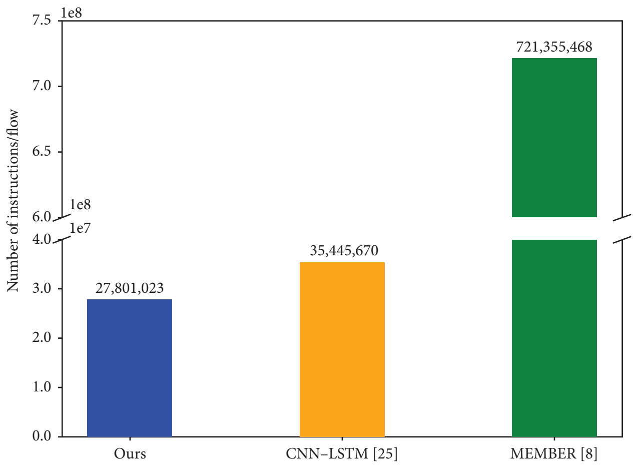


Figure 19: Overall computational performance comparison based on the number of instructions required per flow.

Table 8: Processing time (milliseconds/flow).

	Preprocessing	ML detection	Total
Flow-based	0.73	0.107	0.837
Packet-based	53.36	0.129	53.489

only when necessary. This approach strikes a better balance between performance and computational efficiency.

Moreover, Table 8 provides an overview of the time requirements for preprocessing and ML detection in both the flow-based and packet-based models, revealing that preprocessing is the main bottleneck. Specifically, preprocessing consumes 85% and 99% of the total time in the flow-based and packet-based models, respectively. In the flow-based method, most of the time is spent on feature extraction and statistical calculations, while the packet-based method dedicates the majority of its time to identifying flows and trimming them to a uniform length. Surprisingly, both the flow-based and packet-based methods spend a comparatively small amount of time for ML detection, with the flow-based detection being 20% faster than packet-based detection due to its lower data complexity. The results clearly demonstrate the superiority of the unified approach in detection and its significant improvement in computational resource utilization.

6.5. Unknown Attack Testing. Figure 20 illustrates unknown attack testing accuracy for flow-based and packet-based approaches, covering six types of unknown attack attacks. Remarkably, our model achieved exceptional results, with accuracy above 95% for flow-based and 100% for packet-based detection, showcasing its robustness and strength. These outstanding results can be attributed to the superior generalization capabilities of the CNN model. The CNN's ability to autonomously learn, construct, and interpret traffic patterns within complex networks allows it to detect previously unseen unknown attacks never encountered during

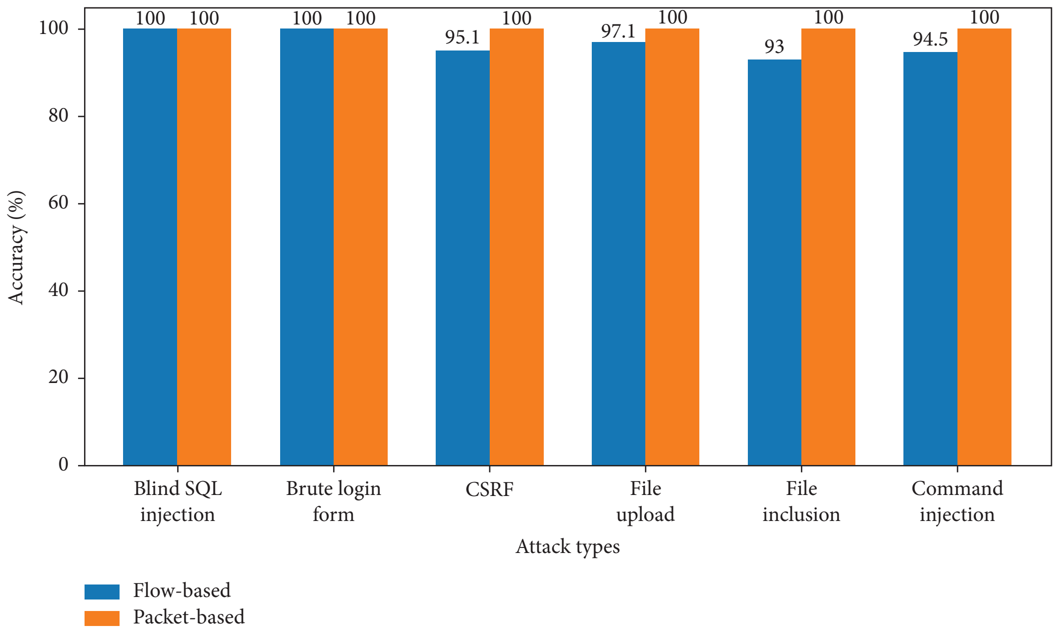


Figure 20: Unknown attack testing result.

training effectively. Furthermore, the ability of the CNN model to recognize unknown attack data reflects its flexibility and adaptability in recognizing and learning from various patterns in network traffic.

7. Conclusions and Future Works

This study presents a unified flow-and-packet ML approach designed for intrusion detection in microservice systems. Combining both flow-based and packet-based techniques, this unified model utilizes flow-based detection as the first defensive layer for shorter flows. Conversely, it uses packet-based detection for longer flows that require more substantial computation. When flow-based detection results in uncertain outcomes below a defined threshold, the data are flagged as potentially falsely detected and redirected to the packet-based detection system for further verification.

This new approach's strength lies in its ability to correct misclassified data using the predetermined threshold. Moreover, it enhances the effectiveness and efficiency in identifying malicious traffic in a microservice system, expedites real-time detection, and holds the capability to detect unknown attacks. For generating datasets, this study adopts a systematic approach to automate the generation of both benign and attack traffic. It involves recording the generated traffic, processing it, and feeding it into our CNN models for both flow- and packet-based detection.

The experimental results demonstrate that this unified approach can achieve a 100% $F1$ score and recall, which is superior to using only flow or packet-based detection. Furthermore, it offers a $24\times$ reduction (approximately 1.38 orders of magnitude) in computational resource usage compared to using only the packet-based approach. This unified approach also proves to be robust against unknown attacks, boasting a 100% accuracy rate in detecting such attacks.

Future work could include evaluating the proposed unified method on real-world microservice systems to validate its effectiveness and scalability in live intrusion detection. This could extend to investigating the impact of different network types and microservice architectures on performance, as well as exploring its adaptability for detecting anomalies in more complex or heterogeneous environments.

Another area for future exploration is optimizing the grey zone threshold tuning process. While our current approach uses exhaustive search, we recognize that alternative optimization techniques, such as genetic algorithms or reinforcement learning, could provide a more adaptive and computationally efficient solution. These methods have the potential to dynamically adjust thresholds based on real-time feedback, improving both detection accuracy and resource efficiency. Integrating such optimization strategies could further enhance the scalability and automation of our framework.

Additionally, future research could focus on real-world deployment and evaluation of the proposed system to further validate its practicality and robustness in operational environments. This includes assessing performance under live traffic conditions, scalability across different infrastructures, and integration with existing security tools.

Data Availability Statement

The data that support the findings of this study are available on request from the first author. The data are not publicly available due to privacy or ethical restrictions.

Conflicts of Interest

The authors declare no conflicts of interest.

Funding

This research was supported by the Industrial Technology Research Institute (ITRI), Taiwan. No funding was received for this manuscript.

Endnotes

[1]https://github.com/nycu-hsl/unified-flow-packet-ids.

References

[1] M. Agiwal, A. Roy, and N. Saxena, “Next Generation 5G Wireless Networks: A Comprehensive Survey,” *IEEE Communications Surveys & Tutorials* 18, no. 3 (2016): 1617–1655, https://doi.org/10.1109/COMST.2016.2532458.

[2] A. Rezaei Nasab, M. Shahin, S. A. Hoseyni Raviz, P. Liang, A. Mashmool, and V. Lenarduzzi, “An Empirical Study of Security Practices for Microservices Systems,” *Journal of Systems and Software* 198 (2023): 111563, https://doi.org/10.1016/j.jss.2022.111563.

[3] J. Liu, X. Song, Y. Zhou, et al., “Deep Anomaly Detection in Packet Payload,” *Neurocomputing* 485 (2022): 205–218, https://doi.org/10.1016/j.neucom.2021.01.146.

[4] G. Marn, P. Casas, and G. Capdehourat, “RawPower: Deep Learning Based Anomaly Detection from Raw Network Traffic Measurements,” in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos. SIGCOMM ’18* (Budapest, Hungary: Association for Computing Machinery, 2018), 75–77, https://doi.org/10.1145/3234200.3234238.

[5] W. Wang, Y. Sheng, J. Wang, et al., “HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection,” *IEEE Access* 6 (2018): 1792–1806, https://doi.org/10.1109/ACCESS.2017.2780250.

[6] M. Lotfollahi, M. Jafari Siavoshani, R. S. H. Zade, et al., “Deep Packet: A Novel Approach for Encrypted Traffic Classification Using Deep Learning,” *Soft Computing* 24.4 (2020): 1999–2012, https://doi.org/10.1007/s00500-019-04030-2.

[7] K. Miyamoto, M. Iida, C. Han, T. Ban, T. Takahashi, and J. Takeuchi, “Consolidating Packet-Level Features for Effective Network Intrusion Detection: A Novel Session-Level Approach,” *IEEE Access* 11 (2023): 132792–132810, https://doi.org/10.1109/ACCESS.2023.3335600.

[8] J. Lan, X. Liu, B. Li, J. Sun, B. Li, and J. Zhao, “MEMBER: A Multi-Task Learning Model with Hybrid Deep Features for Network Intrusion Detection,” *Computers & Security* 123 (2022): 102919, https://doi.org/10.1016/j.cose.2022.102919.

[9] G. Fox and R. V. Boppana, “Detection of Malicious Network Flows With Low Preprocessing Overhead,” *Network* 2, no. 4 (2022): 628–642, https://doi.org/10.3390/network2040036.

[10] K. He, D. D. Kim, and M. R. Asghar, “Adversarial Machine Learning for Network Intrusion Detection Systems: A Comprehensive Survey,” *IEEE Communications Surveys & Tutorials* 25, no. 1 (2023): 538–566, https://doi.org/10.1109/COMST.2022.3233793.

[11] A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, “Machine Learning and Deep Learning Approaches for CyberSecurity: A Review,” *IEEE Access* 10 (2022): 19572–19585, https://doi.org/10.1109/access.2022.3151248.

[12] Y. Zhou, H. Shi, Y. Zhao, W. Gao, and W. Zhang, “Encrypted Network Traffic Identification Based on 2D-CNN Model,” in *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2021), 238–241, https://doi.org/10.23919/APNOMS52696.2021.9562636.

[13] W. Niu, Z. Zhuo, X. Zhang, X. Du, G. Yang, and M. Guizani, “A Heuristic Statistical Testing Based Approach for Encrypted Network Traffic Identification,” *IEEE Transactions on Vehicular Technology* 68, no. 4 (2019): 3843–3853, https://doi.org/10.1109/TVT.2019.2894290.

[14] Z. Bu, B. Zhou, P. Cheng, K. Zhang, and Z.-H. Ling, “Encrypted Network Traffic Classification Using Deep and Parallel Network-In-Network Models,” *IEEE Access* 8 (2020): 132950–132959, https://doi.org/10.1109/ACCESS.2020.3010637.

[15] R. Vinayakumar, K. P. Soman, P. Poornachandran, and S. Akarsh, “Application of Deep Learning Architectures for Cyber Security,” in *Advanced Sciences and Technologies for Security Applications* (2019).

[16] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, “End-to-End Encrypted Traffic Classification With One-Dimensional Convolution Neural Networks,” in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)* (2017), 43–48, https://doi.org/10.1109/ISI.2017.8004872.

[17] Z. Okonkwo, E. Foo, Q. Li, and Z. Hou, “A CNN Based Encrypted Network Traffic Classifier,” in *ACSW’22* (Brisbane, Australia: Association for Computing Machinery, 2022), 74–83, https://doi.org/10.1145/3511616.3513101.

[18] B. Wang, Y. Su, M. Zhang, and J. Nie, “A Deep Hierarchical Network for Packet-Level Malicious Traffic Detection,” *IEEE Access* 8 (2020): 201728–201740, https://doi.org/10.1109/ACCESS.2020.3035967.

[19] I. Ullah and Q. H. Mahmoud, “A Two-Level Hybrid Model for Anomalous Activity Detection in IoT Networks,” in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)* (2019), 1–6, https://doi.org/10.1109/CCNC.2019.8651782.

[20] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, “A Hierarchical Hybrid Intrusion Detection Approach in IoT Scenarios,” in *GLOBECOM 2020-IEEE Global Communications Conference* (2020), 1–7, https://doi.org/10.1109/GLOBECOM42002.2020.9348167.

[21] M. Ashfaq Khan, Md. Rezaul Karim, and Y. Kim, “A Scalable and Hybrid Intrusion Detection System Based on the Convolutional-LSTM Network,” in *Symmetry 11.4* (2019), https://doi.org/10.3390/sym11040583.

[22] M. Verkerken, L. D’hooge, D. Sudyana, et al., “A Novel Multi-Stage Approach for Hierarchical Intrusion Detection,” *IEEE Transactions on Network and Service Management* 20, no. 3 (2023): 3915–3929, https://doi.org/10.1109/TNSM.2023.3259474.

[23] D. F. Rueda, J. C. Caviedes, and W. Yesid Campo Muñoz, “A Hybrid Intrusion Detection Approach Based on Deep Learning Techniques,” in *Computer Networks, Big Data and IoT*, ed. A. Pasumpon Pandian, X. Fernando, and W. Haoxiang (Singapore: Springer Nature Singapore, 2022), 863–878.

[24] A. Ammar, “Toward Efficient Intrusion Detection System Using Hybrid Deep Learning Approach,” *Symmetry* 14.9 (2022): https://doi.org/10.3390/sym14091916.

[25] S. S. Bamber, A. V. R. Katkuri, S. Sharma, and M. Angurala, “A Hybrid CNN-LSTM Approach for Intelligent Cyber Intrusion Detection System,” *Computers & Security* 148 (2025): 104146, https://doi.org/10.1016/j.cose.2024.104146.

[26] Z. S. Mahdi, R. M. Zaki, and L. Alzubaidi, “Advanced Hybrid Techniques for Cyberattack Detection and Defense in IoT Networks,” in *Security and Privacy* (2024), https://doi.org/10.1002/spy2.471.

[27] Z. Zulfiqar, S. U. R. Malik, S. A. Moqurrab, Z. Zulfiqar, U. Yaseen, and G. Srivastava, “DeepDetect: An Innovative

Hybrid Deep Learning Framework for Anomaly Detection in IoT Networks," *Journal of Computational Science* 83 (2024): 102426, https://doi.org/10.1016/j.jocs.2024.102426.

[28] M. Koca and I. Avci, "A Novel Hybrid Model Detection of Security Vulnerabilities in Industrial Control Systems and IoT Using GCN+LSTM," *IEEE Access* 12 (2024): 143343–143351, https://doi.org/10.1109/ACCESS.2024.3466391.

[29] A. M. Alsaffar, M. Nouri-Baygi, and H. M. Zolbanin, "Shielding Networks: Enhancing Intrusion Detection with Hybrid Feature Selection and Stack Ensemble Learning," *Journal of Big Data* 11, no. 1 (2024): 133, https://doi.org/10.1186/s40537-024-00994-7.

[30] S. Cai, D. Han, X. Yin, D. Li, and C.-C. Chang, "A Hybrid Parallel Deep Learning Model for Efficient Intrusion Detection Based on Metric Learning," *Connection Science* 34, no. 1 (2022): 551–577, https://doi.org/10.1080/09540091.2021.2024509.

[31] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networks," *IEEE Transactions on Network and Service Management* 16, no. 3 (2019): 924–935, https://doi.org/10.1109/TNSM.2019.2927886.

[32] GoogleCloudPlatform, "Google Microservices Demo," https://github.com/GoogleCloudPlatform/microservices-demo.

[33] Y. Gan, Y. Zhang, D. Cheng, et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud Amp; Edge Systems," in *ASPLOS'19* (Providence: Association for Computing Machinery, 2019), 3–18, https://doi.org/10.1145/3297858.3304013.

[34] T. Jain and N. Jain, "Framework for Web Application Vulnerability Discovery and Mitigation by Customizing Rules through ModSe-Curity," in *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)* (2019), 643–648, https://doi.org/10.1109/SPIN.2019.8711673.

[35] Antirez, "Antirez/hping: Hping Network Tool," https://github.com/antirez/hping.

[36] G. Yaltirakli, "Slowloris," in *github.com* (2015), https://github.com/gkbrk/slowloris.

[37] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *International Conference on Information Systems Security and Privacy* (2018).

[38] F. Yudha, Y.-D. Lin, Y.-C. Lai, D. Sudyana, and R.-H. Hwang, "Reproducing ATTCK Techniques and Lifecycles to Train Machine Learning Classifier," in *IEEE Network* (2025), 1, https://doi.org/10.1109/MNET.2025.3551333.

[39] Z. Aouini and A. Pekar, "NFStream: A Flexible Network Data Analysis Framework," *Computer Networks* 204 (2022): 108719, https://doi.org/10.1016/j.comnet.2021.108719.

[40] R.-H. Hwang, M.-C. Peng, C.-W. Huang, Po-C. Lin, and V.-L. Nguyen, "An Unsupervised Deep Learning Model for Early Network Traffic Anomaly Detection," *IEEE Access* 8 (2020): 30387–30399, https://doi.org/10.1109/ACCESS.2020.2973023.

[41] L. Mohammadpour, T. Chaw Ling, C. Sun Liew, and A. Aryanfar, "A Survey of CNN-Based Network Intrusion Detection," in *Applied Sciences* (2022), 8162, https://doi.org/10.3390/app12168162.

[42] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-Generation Hyperparameter Optimiza-tion Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD'19* (Anchorage, AK, USA: Association for Computing Machinery, 2019), 2623–2631, https://doi.org/10.1145/3292500.3330701.