A-NIDS: Adaptive Network Intrusion Detection System Based on Clustering and Stacked CTGAN

Chao Zha, Zhiyu Wang, Yifei Fan, Bing Bai, Yinjie Zhang, Sainan Shi, and Ruyun Zhang✉

***Abstract*—Intrusion detection systems (IDS) are crucial tools for detecting anomalous network traffic in cybersecurity. In recent years, significant progress has been made in applying artificial intelligence to IDS. However, existing research often assumes that training and testing data are static and identically distributed, whereas in reality, data drift is inevitable. Moreover, to enhance model versatility and detection performance, models have become increasingly complex, posing challenges to real-time deployment. To address these challenges, we propose an adaptive network intrusion detection system named A-NIDS, consisting of a main task and two bypass tasks. The main task is to develop a fully connected and shallow network with strong detection performance and real-time capability. The first bypass task is a clustering model that helps the main task detect data drift in an unsupervised manner. The second bypass task is a generation model to generate old data to address catastrophic forgetting in new model iterations and the storage cost issue caused by accumulating old data. We conduct extensive experiments on the CICIDS-2017 and CSE-CICIDS-2018 datasets, demonstrating the superior performance of A-NIDS on new and old data. Furthermore, our detection module achieves a detection latency of 5 microseconds, highlighting its suitability for real-time applications. All the related code is publicly available at: https://github.com/ids-sec-hub/A-NIDS.**

***Index Terms*—Intrusion detection, adaptive, data drift, clustering, generation, latency.**

I. Introduction

INTRUSION DETECTION SYSTEMS play a crucial role in network security by helping organizations prevent unauthorized access, malware, and various network threats [1]–[3]. Traditional IDS commonly employ predefined rules or features to detect anomalies or malicious activities [4]–[9]. Feature-based IDS depend on recognized attack features, such as specific patterns in network traffic or malware signatures. In contrast, rule-based IDS detects potential intrusion behaviors through predefined rules, such as traffic on specific ports or the use of particular protocols. Recent advancements have shifted IDS technology from feature-based and rule-based techniques to machine learning-based methodologies.

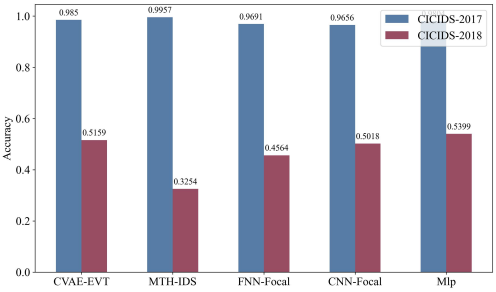


Fig. 1. Phenomenon of data drift. We select four different state-of-the-art methods and MLP, and train them on the CICIDS-2017 dataset, and then test them on the CICIDS-2018 dataset to evaluate their ability to handle data drift.

Machine learning has significantly advanced the field of network intrusion detection systems (NIDS) [10]–[12]. Compared to traditional feature-based or rule-based methods, machine learning offers superior capabilities to handle new and unknown attack patterns, reduce false alarms, and improve accuracy and efficiency [13]. Initially, decision trees, random forests, support vector machines and deep learning algorithms such as autoencoders dominated the early applications of machine learning in IDS [14]–[19]. These algorithms were used to extract feature information from traffic data for classification tasks. As research progressed, efforts focused on addressing issues such as data imbalance and vulnerability to adversarial attacks [20]–[23]. Consequently, methods such as data augmentation and frequency domain transformation were proposed to overcome these challenges.

Existing research has predominantly focused on these narrow-scope studies, assuming that the network environment exhibits a static data distribution over a short time frame, which may not align with the dynamic nature of real-world scenarios. Traffic feature distributions may change due to factors such as system services and business modifications in real-world scenarios, resulting in inconsistency with the feature distributions of old data - a phenomenon known as data drift [24]. Specifically, the application of machine learning in IDS faces challenges in adapting to data drift and generalizing models. We train and test four state-of-the-art methods [16], [25], [26] and Multilayer Perceptron (MLP) on two publicly

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received April 19, 2021; revised August 16, 2021 (Corresponding author: Ruyun Zhang).

Chao Zha is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; the Intelligent Computing Infrastructure Innovation Center, Zhejiang Lab, Hangzhou 311500, Zhejiang, China; the University of the Chinese Academy of Sciences, Beijing 100049, China; and the School of Intelligent Science and Technology, Hangzhou Institute for Advanced Study, UCAS, Hangzhou 311500, Zhejiang, China. (email: zhachao21@mails.ucas.ac.cn).

Zhiyu Wang, Yifei Fan, Bing Bai, Yinjie Zhang, and Ruyun Zhang are with the Intelligent Computing Infrastructure Innovation Center, Zhejiang Lab, Hangzhou 311500, Zhejiang, China (email: wangzhy@zhejianglab.org; yffan@zhejianglab.org; baibing@zhejianglab.org; zyj19961126@zhejianglab.org; zcor2021@gmail.com).

Sainan Shi is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; and the Intelligent Computing Infrastructure Innovation Center, Zhejiang Lab, Hangzhou 311500, Zhejiang, China. (email: shisainan22@mails.ucas.ac.cn).

available datasets, CICIDS-2017 [27], [28] and CICIDS-2018 [28], [29]. The results, as shown in Fig. 1, reveal a sharp decline in accuracy for all methods when tested on datasets with data drift. This suggests that these methods are inadequate to address the problem of data drift.

Although retraining with new data can indeed yield good performance on the new data, it often suffers from catastrophic forgetting of old data. Another brute-force solution is to retain the old data and train a new model with new data to adapt to the new network environment. However, continuing to retain old data incurs significant storage costs. Furthermore, the real-time requirements of IDS pose major performance barriers to machine learning algorithms, limiting their practical application in real-world scenarios [8], [23], [30].

To overcome these challenges, we present A-NIDS, an adaptive network intrusion detection system. The method proposed in A-NIDS addresses the aforementioned challenges: (1) adaptively detecting whether the network data distribution has drifted; (2) addressing the problem of catastrophic forgetting and the cost of continuously storing old data; and (3) addressing the problem of real-time detection capability in NIDS. Finally, we conduct a series of experiments and analyzes on two publicly available datasets, CICIDS-2017 and CSE-CICIDS-2018 [28], demonstrating that our approach significantly outperforms existing methods in these problems.

The contributions of this paper are as follows.

- We propose a sliding data window to capture traffic data and use a clustering model along with outlier analysis to adaptively detect data drift in the traffic window.
- We train a stacked Conditional Tabular Generative Adversarial Network (CTGAN) model to generate data similar to the distribution of old data, addressing the storage overhead of old data.
- We jointly train a detection model using a shallow fully connected network with both new and generated data, solving the catastrophic forgetting problem, learning new knowledge, and achieving good real-time performance.

The rest of the paper is organized as follows. Section II introduces the background of the generation model. Section III describes our threat model. Section IV presents the motivations and challenges of the current NIDS. In Section V, we describe the detailed design of A-NIDS. In Section VI, we conduct the theoretical analysis of two critical hyperparameters in the Adative Module. In Section VII, we experimentally evaluate the performance of our method. Section VIII discusses the potential adversarial attacks that the system may face and some feasible defense methods. Section IX discusses the limitations of our proposed method as well as future work. Section X reviews related work. Finally, we conclude this paper in Section XI.

II. Background

A. Generation model

Generation models learn the distribution of training data and utilize this acquired knowledge to generate new data samples reminiscent of the training dataset, which has wide applications in many fields, including natural language processing, image synthesis in computer vision and audio processing. Existing generation models include variable auto-encoder (VAE) [31], diffusion probabilistic models (DDPM) [32], and Generative Adversarial Networks (GAN) [33]. VAE maximizes the lower bound of the log-likelihood of the observed data during training, enabling the model to capture latent data structures and learn how to generate data. DDPM generates data via an evolutionary process, gradually diffusing noise into the data space to create new samples. GAN consists of a generator that produces fake data samples and a discriminator that distinguishes between fake and real samples. Through adversarial training, the generator and the discriminator compete with each other in an adversarial setting to produce realistic data samples.

Compared to GAN, VAE-generated samples may lack realism, as they tend to lose information during training and easily get caught in local optima, as shown in some studies [34]. DDPM produces high-quality samples by gradually removing noise to generate data. However, its inference process is slow, requiring more iterations during inference and longer overall training times. GAN can rapidly generate high-quality data and produce samples with diverse distributions.

We leverage the generation model to produce old data, allowing it to be used alongside new data for the training detection model, thereby achieving superior detection performance on new data without catastrophically forgetting old data. Furthermore, the generation model can address issues related to storage overhead caused by the continuous accumulation of historical data, as well as privacy concerns during model migration. Although generated data may lose a small portion of information compared to original real data, leading to some forgetting, this trade-off is acceptable. In this paper, we use CTGAN [35] to produce old data, as it handles mode collapse issues well. In addition, we have made several improvements based on CTGAN.

B. CTGAN

CTGAN[1] [35] is a GAN-based method designed to capture the distribution of tabular data and generate rows that align with this distribution. Training an effective GAN for tabular data is challenging due to the diverse nature of numerical data, which can be continuous or discrete. Continuous values often follow a Variational Gaussian Mixture (VGM) model, which can lead to the issue of mode collapse. The uneven distribution of discrete values may result in imbalanced generated data. To address these challenges, two adjustment strategies are proposed in CTGAN.

Mode-Specific Normalization (MSN). For discrete features, MSN is represented using one-hot vectors. For continuous features, unlike *min-max normalization*, MSN uses a VGM model [36] to estimate the number of modes, denoted m_i. As shown in Fig. 2, VGM identifies three modes ($m_i = 3$), namely η_1, η_2 and η_3. The learned VGM is $P_{C_i}(c_{i,j}) = \sum_{k=1}^{3} \mu_k N(c_{i,j}; \eta_k, \phi_k)$, where η_k and ϕ_k represent the weights and standard deviations of the modes, respectively. Furthermore, the probability densities ρ_1, ρ_2, ρ_3 for $c_{i,j}$ of each mode are computed as $\rho_k = \mu_k N(c_{i,j}; \eta_k, \phi_k)$.

[1]CTGAN is open-sourced at https://github.com/DAI-Lab/CTGAN

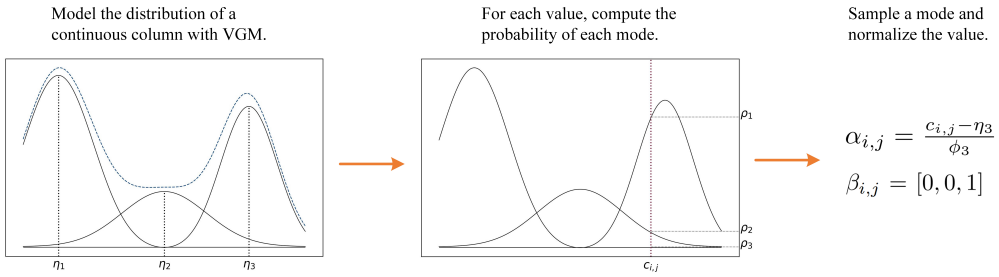


Fig. 2. An example of Mode-Specific Normalization.

Finally, a mode is sampled from the given probability densities and the sampled mode is used to normalize the values. For example, selecting the third mode from the given ρ_1, ρ_2, and ρ_3 is represented by a one-hot vector $\beta_{i,j} = [0, 0, 1]$, and a scalar $\alpha_{i,j} = \frac{c_{i,j}-\eta_3}{\phi_3}$ represents the value within this mode. So, the representation of a row becomes a concatenation of continuous and discrete columns, as follows:

$$r_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \ldots \oplus \alpha_{N_c,j} \oplus \beta_{N_c,j} \oplus d_{1,j} \oplus \ldots \oplus d_{N_d,j}, \quad (1)$$

where $d_{i,j}$ is the one-hot representation of a discrete value, N_c and N_d denote the numbers of columns for continuous and discrete features, respectively.

Training-by-Sampling. Traditional GAN generator receives a vector sampled from the standard multivariate normal distribution and obtains a deterministic transformation mapping it to the data distribution. However, this approach does not consider the imbalance in the categorical columns. GTGAN addresses this issue through three key elements: *Conditional Vector*, *Generator Loss*, and *Sampling Training*. *Conditional Vector* is formed by concatenating N_d mask vectors, i.e., $cond = m_1 \oplus ... \oplus m_{N_d}$, where $m_i = [m_i^{(k)}]$, $m_i^{(k)} = 1$ if k is the selected category and 0 otherwise. During *Sampling Training*, a zero-filled mask m_i is first created for each discrete column D_i, then randomly select a column i, and the frequency of each category is computed to construct a probability mass function (PMF). The PMF is used to randomly selects a category k, and a *Conditional Vector cond* is constructed based on the selected i and k. Finally, the conditional generator takes *cond* as input and generates data. To ensure that the generator can produce data based on the given conditions, a cross-entropy loss function between the *Conditional Vector* and the generated data is introduced.

Consequently, CTGAN can learn the overall distribution of the training data and effectively capture the general shape of each feature's distribution. However, CTGAN often fails to capture significant shared information between features and sometimes overlooks correlations between features and other critical aspects.

III. Threat Model

We consider an intrusion detection scenario in a dynamic network environment. Specifically, NIDS is used to monitor network traffic with the objective of detecting potential attacks, unauthorized access, or anomalous behavior, thus safeguarding computer systems and networks against threats. Typically, NIDS is deployed at critical network nodes, such as switches, routers, or alongside firewalls. NIDS faces a wide range of complex threats, including known traditional attacks such as malware, denial-of-service (DoS) attacks, distributed denial-of-service (DDoS) attacks, and brute-force attempts. As attack techniques continue to evolve, NIDS must also contend with increasingly sophisticated unknown threats, such as zero-day exploits and hybrid attacks like botnets. Adversaries exploit the dynamic characteristics of network traffic, employing tactics such as altering attack patterns, disguising malicious activities as normal traffic, or leveraging uncovered attack vectors to evade detection.

Moreover, the distributions of the features of the network traffic change over time due to changes in user behavior, updates in normal network traffic, or adaptations to attacker strategies – a phenomenon called data drift. This drift causes the detection performance of traditional DL-NIDS to degrade over time, leading to increased false positives or even detection failure for drifted data. From the perspective of robust IDS, the objectives are twofold: 1) Adaptively identify the occurrence of data drift in an unsupervised manner, thereby avoiding the burdensome task of repeated data labeling and unnecessary model updates. 2) Effectively update the model without access to historical data, ensuring accurate detection of drifted data while avoiding catastrophic forgetting of previous knowledge.

IV. Motivations and Challenges

In this section, we will discuss our motivations and the challenges faced by NIDS.

A. What is the primary elements of NIDS?

False Positive Rate (FPR) *is defined as the ratio of normal traffic incorrectly identified as malicious*. Normal traffic far exceeds malicious traffic in real-world network environments. A high FPR can lead to significant interception of normal traffic, affecting the availability of protected system services. Each percentage point increase in the FPR can result in a large number of affected users, with impacts that are difficult to quantify. Although normal traffic is typically classified under the same label, it should actually be divided into multiple subtypes, which will increase with data drift. Failure to capture data drift in a timely manner can result in these new types of normal traffic being misidentified as abnormal traffic, further increasing the FPR. Such NIDS that intercept a large

amount of normal access will be greatly limited in practical deployment. Therefore, ensuring a low FPR is one of the key elements of NIDS.

False Negative Ratio (FNR) *is defined as the ratio of malicious traffic identified as normal traffic*. When NIDS encounters data drift, new attack traffic can be considered zero-day attacks. Existing research on zero-day attack detection has demonstrated limited detection performance. In our experiments and previous research [26], we have found that most of these zero-day attacks are misclassified as normal traffic, significantly increasing the FNR and thus reducing the effectiveness of NIDS. Furthermore, identified zero-day attacks are not accurately classified as any specific attack behavior, hindering the deployment of precise defensive measures. The timely detection of this data drift phenomenon and the adaptation of the models accordingly would represent a significant advancement in NIDS.

B. Challenges in current DL-NIDS

Previous Deep Learning-based NIDS (DL-NIDS) have generally focused solely on metrics such as detection accuracy, while overlooking several other critical challenges. Here, we outline the key issues currently faced by DL-NIDS.

Ch1: *Data drift.* Previous DL-NIDS methods typically assumed a static data distribution [26]. However, this assumption is not applicable in real-world deployment environments. Although the distribution may remain static for a limited period, continuous changes in the network environment inevitably lead to changes in the distribution of traffic. Consequently, traditional DL-NIDS methods, which largely overlook this factor, encounter significant limitations in real-world deployments.

Ch2: *Model reliability.* Based on the assumption of a static traffic distribution, previous DL-NIDS approaches may provide reliable detection results for in-distribution samples. However, the detection results for the out-of-distribution (OOD) samples are not reliable, as they deviate from a presumed distribution. With the occurrence of data drift, the emergence of OOD samples becomes inevitable, further undermining the model's detection reliability.

Ch3: *Catastrophic forgetting of old knowledge.* Model updating is a significant issue in deep learning, particularly in dynamic environments. As new knowledge is learned, catastrophic forgetting of old knowledge may occur, leading to poor detection performance when old samples reappear. Such outcomes are undesirable, highlighting the importance of retaining both new and old knowledge. While some forgetting of prior knowledge can be tolerated to prevent overly complex model structures, catastrophic forgetting is unacceptable in NIDS scenarios.

Ch4: *Real-time performance.* Relying solely on a complex model to address the myriad challenges encountered in NIDS may seem viable in theory. However, this approach presents a significant drawback: it does not meet real-time performance requirements. Ignoring real-time considerations during practical deployment limits the feasibility and usability of such applications.

C. How to adaptively detect data drift?

Data drift can be explained as a significant difference between the new data distribution experienced during the model's operation over time and the old data distribution used during training, leading to a decline in the model's detection accuracy and performance. To better illustrate this issue, we represent a dynamic process as shown in Fig. 3. The initial state is considered the old data distribution, denoted as $D_{\text{old}} = \{X_1, X_2, \ldots, X_d\}$. As time progresses, a new state is considered as the new data distribution, denoted as $D_{\text{new}} = \{X_1, X_2, \ldots, X_{d'}\}$. If D_{new} is inconsistent with the distribution of D_{old} and the model fails in D_{new}, the data drift is considered to have occurred with respect to D_{old}.

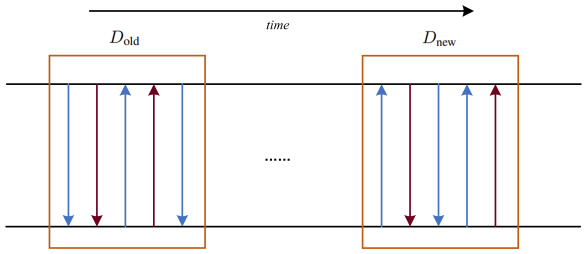


Fig. 3. The overview of data drift.

Many existing methods can be used to measure differ between two data distributions. We have studied several widely used methods, and analyzed their internal principles, and then identified specific challenges in applying them to traffic data drift detection. The subsequent sections provide a detailed discussion of three representative methods.

Maximum Mean Discrepancy (MMD). As shown in Eq. (2), given two data distributions D_{old} and D_{new}, MMD [37] measures the distance between the two probability distributions and is often used to assess the similarity or dissimilarity between two datasets.

$$\text{MMD}(D_{old}, D_{new}) = \left\| \mathbb{E}_{x \sim D_{old}}[\phi(x)] - \mathbb{E}_{y \sim D_{new}}[\phi(y)] \right\|^2, \tag{2}$$

where ϕ denotes the feature map function that maps data from the input space to a feature space, and $\mathbb{E}_{x \sim D}[\cdot]$ represents the expectation with regard to the distribution D.

Kullback-Leibler Divergence (KLD). As shown in Eq. (3), KLD [38] measures the divergence between two probability distributions. Let P_{old} and P_{new} represent the probability density functions of the data distributions D_{old} and D_{new}, respectively. $P_{old}(i)$ and $P_{new}(i)$ denote the probabilities of distributions P_{old} and P_{new} at data point i. The KLD is zero only when $P_{old}(i) = P_{new}(i)$, indicating identical distributions.

$$D_{KL}(P_{old} \mid\mid P_{new}) = \sum_i P_{old}(i) \log \left(\frac{P_{old}(i)}{P_{new}(i)} \right) \tag{3}$$

Adversarial Validation (AVD). As shown in Fig. 4, the AVD [39] combines the data distributions D_{old} and D_{new} into a unified data distribution D, assigning labels 0 and 1 to indicate samples from D_{old} and D_{new}, respectively. Subsequently, a binary classifier f is trained on D to achieve optimal classification performance. If the classifier does not

distinguish effectively between the two data distributions, it indicates a significant similarity between them.

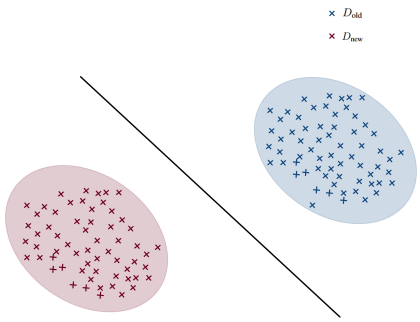


Fig. 4. The overview of Adversarial Validation.

In summary, the above methods only measure the differences between the overall distributions and cannot capture local distribution differences. Moreover, MMD depends on the selection of the feature mapping function, and different feature mapping functions lead to different distance measurement results. Furthermore, after selecting the feature function, proper parameter tuning is essential to achieve better performance. However, we aim to avoid storing old data, which makes parameter tuning impractical. Therefore, in certain cases, these three methods may not adequately describe the detailed differences in the distributions.

However, our objective is to adaptively detect the data drift between D_{old} and D_{new} on different labels. While the overall data distribution may unchanged, local shifts in specific label distributions can occur and necessitate a prompt response. Notably, D_{new} comprises data from multiple labels, but we do not know their specific labels at this stage. Consequently, this problem is inherently unsupervised and the aforementioned methods are not suitable. We propose a clustering-based unsupervised approach to address this, which will be detailed in Section V-C.

V. Our Method

In this section, we present the methodology of **A-NIDS**, the overall architecture and the design of three main modules in **A-NIDS**. Fig. 5 illustrates these three modules of our A-NIDS, detailing their components and interrelationships.

A. Overview

To address the issues of data drift and real-time performance faced by traditional DL-NIDS in practical deployment, our framework employs a shallow neural network for the main task of NIDS, supplemented by two bypass tasks to improve detection reliability. The shallow neural network ensures improved real-time performance, but bring challenges such as invisibility to data drift and catastrophic forgetting of old data when learning new knowledge. We introduce two bypass tasks, the *Adaptive Module* and the *Generation Module*, to mitigate these problems. The *Adaptive Module* adaptively detects data drift based on an unsupervised clustering model. The *Generation Module* generates synthetic old data, which are combined with new data to train the main model, enabling the learning of new knowledge while mitigating catastrophic forgetting of old knowledge. Compared to the main task, bypass tasks have lower real-time requirements.

In general, our approach involves two distinct data flows. The flow of old data is represented by the solid lines in Fig. 5. The old data are used to train a classifier, a clustering model, and a generation model. The classifier is used to detect old data, the clustering model is applied to continuously monitor data drift, and the generation model is used to produce old data when data drift occurs. The flow of new data is represented by the dashed lines in Fig. 5. The new data are first captured by a sliding window mechanism and then analyzed by the clustering model to detect the occurrence of data drift. When data drift is detected, the generation model produces old data, which is combined with the new data to retrain the classifier. The newly trained classifier subsequently replaces the old classifier.

B. Main Task: Detection Module

As described in **Ch4**, applying a complex model to address the various challenges in DL-NIDS (introduced in Section IV-B) presents a promising solution. However, it comes with a significant drawback: a lack of real-time responsiveness. Hence, we decompose the challenges faced by DL-NIDS and identify the real-time problem as the primary focus, which has been well developed in previous research. We choose this as our *Main Task*.

To meet real-time requirements, we design the network architecture as a fully connected and shallow network. Shallow networks typically have fewer parameters and computational requirements, resulting in faster training and inference speeds as well as lower demands on computing resources and memory. Before input vectors enter the model, we preprocess the data using the traditional *Min-Max Normalization* method to facilitate faster convergence of the model.

C. Bypass Task 1: Adaptive Module

As described in Section IV-C, data drift detection is an unsupervised problem. To address this, we design an *Adaptive Module* based on a window mechanism and a clustering algorithm.

As shown in Fig. 6, the window mechanism maintains a sliding window (introduced in **Section IV-C**) that stores a certain amount of data from the network, continuously updating with the latest data over time. At regular intervals, the data within the window is evaluated by a clustering model to detect data drift. If data drift occurs, the module sends an activation signal to both the *Generation Module* and the *Detection Module*, prompting them to update the classifier.

As shown in Fig. 7, the clustering model analyzes the correlations between samples and groups them into multiple clusters, thereby revealing the intrinsic structure of the data through automatic clustering. The advantage of using a clustering algorithm is that it allows us to observe local distribution patterns even without labels.

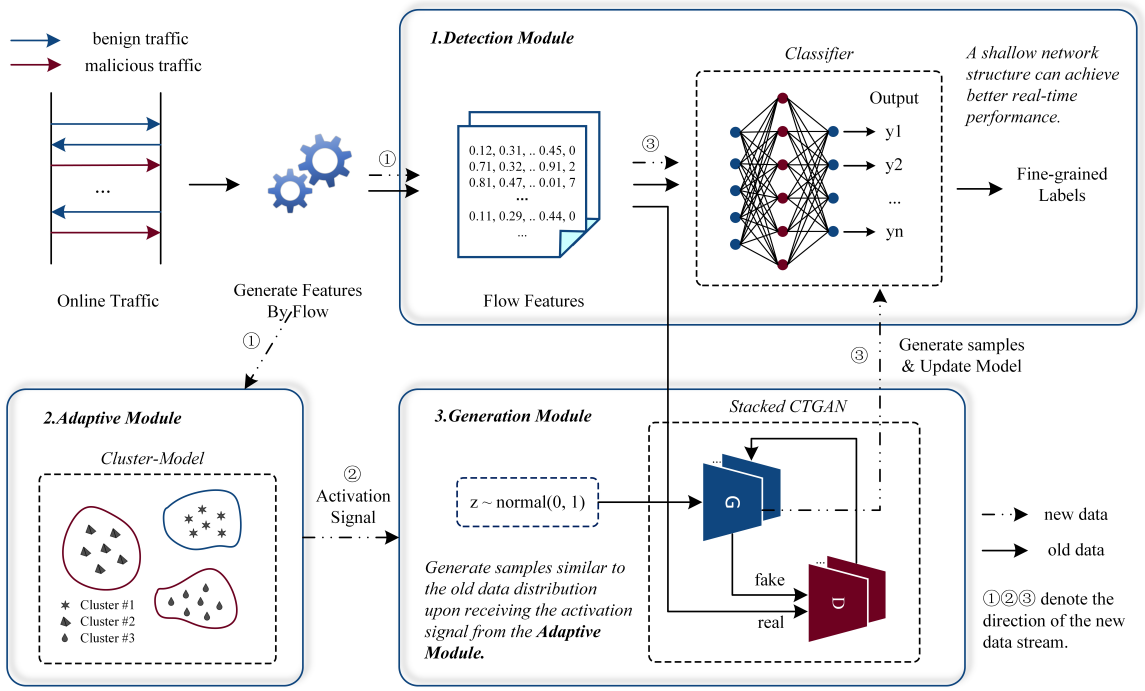


Fig. 5. The overview of A-NIDS. It consists of three main modules: *Detection Module*, *Adaptive Module*, and *Generation Module*. The *Detection Module* consists of several layers of fully connected networks, characterized by simplicity and real-time detection. The *Adaptive Module* is a clustering model that checks whether there is drift in the detection data for each new traffic window. The *Generation Module* is a stacked CTGAN model that, when it receives the activation signal from the *Adaptive Module*, trains a new model with the old data and the new data from the current traffic window to adapt to the new environment. Solid lines in the figure represent old data stream, dashed lines represent new data stream, and ① - ③ indicate the direction of the new data stream.

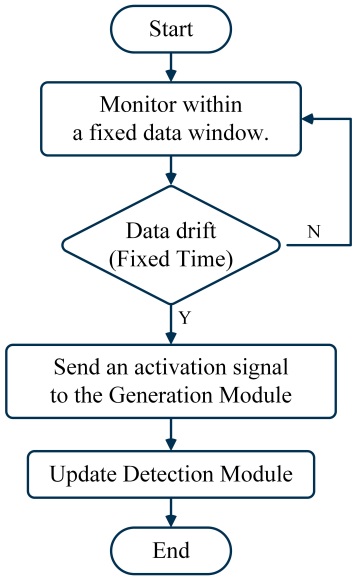


Fig. 6. The overview of window mechanism.

Clustering algorithms can be implemented in various ways [40]. We employ a partition-based approach, such as K-Means [41], which assigns samples to K clusters iteratively optimizing the objective function, minimizing the sum of squared distances between the samples and their respective cluster centers. We denote the clustering model as $f_C : X \to [0, 1]^K$, where C denotes the cluster center, X represents the traffic space, and $Y = \{1, 2, ..., K\}$ represents the pseudo-labels. Let $L(X_i) = \|X_i - C(Y_{X_i})\|^2$ denote the distance between a sample X_i and the center $C(Y_{X_i})$ of its assigned cluster. Two evaluation metrics involved in our algorithm can be defined as follows.

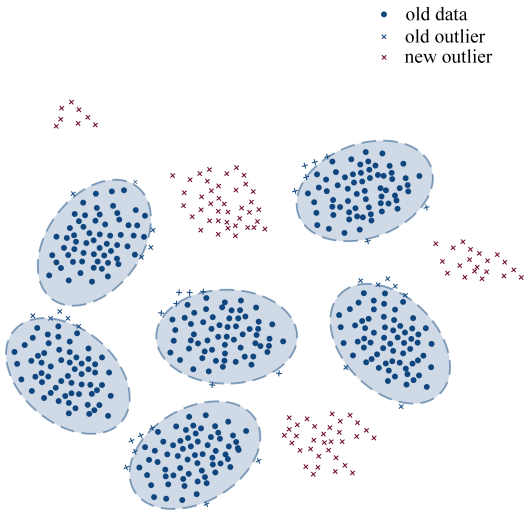


Fig. 7. The overview of clustering algorithm.

1) The confidence interval $[0,\ CI]$ defines the range within which the clustering results of all samples are considered reliable; samples outside this interval are considered outliers.

$$CI = \mathbb{E}_{X_j \sim D_{old}}[L(X_j)] + \alpha \cdot \mathbb{S}_{X_j \sim D_{old}}[L(X_j)], \quad (4)$$

where $\mathbb{E}_{X_j \sim D_{old}}[L(X_j)]$ and $\mathbb{S}_{X_j \sim D_{old}}[L(X_j)]$ represent the expectation and the standard deviation of the distance from the sample X_j in the old distribution D_{old} to the cluster center $C(X_j)$, respectively, and α denotes a hyperparameter.

2) The outlier ratio (OR) is defined as the weighted ratio of the number of outliers in the new distribution D_{new} to the number of outliers in the old distribution D_{old}.

$$OR = \frac{\frac{1}{d'}\sum_{k}^{d'} \mathbb{I}\{L(X_k) \notin [0, CI]\}}{\frac{1}{d}\sum_{j}^{d} \mathbb{I}\{L(X_j) \notin [0, CI]\}}, \tag{5}$$

where d' denotes the number of samples in the new distribution D_{new}, d denotes the number of samples in the old distribution D_{old}, and $\mathbb{I}\{A\} = 1$ if and only if the condition A is true.

Consequently, when OR exceeds a threshold T, we define that a data drift has occurred in the new data distribution D_{new}, and this drift has occurred in multiple local distributions. After detecting data drift, The *Generation Module* will be activated to generate old data to train the new classifier. K and α are two critical hyperparameters in this module, and their values directly influence its performance. A detailed analysis of them is presented in Section VI.

D. Bypass Task 2: Generation Module

We designed a *Generation Module* to produce old traffic data, aiming to prevent catastrophic forgetting in the new classifier while accurately identifying new data. Moreover, this module addresses the storage and privacy issues associated with the continuous accumulation of old data. The *Generation Module* is based on CTGAN; however, as discussed in Section II-B, CTGAN does not capture the correlation between features and labels. In other words, while the generated data ensures overall distributional similarity, it cannot guarantee similarity in the label-related local distributions.

CTGAN abandons the traditional *Min-Max Normalization* for data normalization and instead adopts MSN. This approach is based on the assumption that the distribution of data features follows VGM. By adopting this approach to construct a generation model for old data, we essentially make the assumption that all our data conform to the same VGM on the same feature data. However, this assumption is not valid, the same feature data corresponding to different labels may follow different VGM models.

To address this issue, we propose a **Stacked-CTGAN**, which individually fits a VGM to the feature data of different labels for data normalization. This approach allows us to capture the distribution characteristics of the data with different labels more effectively, thus improving the quality of the generated data.

In conclusion, the *Generation Module* can generate synthetic data that closely resemble real datasets, which can be used to train machine learning model without the risk of leaking sensitive information. We use a generation model to simulate the distribution of old data, allowing us to create old data during the training of the new model and combine it with new data for training.

VI. Theoretical Analysis

In this section, we theoretically analyze the two critical hyperparameters, K and α, in the *Adaptive Module* to assist in accurately selecting the appropriate values for practical deployment.

A. The Number of Clusters K in Clustering Algorithms

As mentioned above, the traffic feature data is fitted to a VGM. If K is set to the number of true labels L, the fit of the model is poorer compared to setting K at a value several times greater than L. We aim for each cluster to fit a Gaussian distribution, which means that after clustering, the samples within each cluster follow a Gaussian distribution. In this case, our assumption is reasonable: data with the same label exhibit multiple distributions in data features, and this multi-distribution phenomenon can be captured by increasing the number of clusters. However, selecting a very large value for K is not optimal; when K is too large, each cluster may contain only a few samples, leading to overfitting and a lack of generalizability in the model.

B. The Confidence Coefficient α for Clustering Algorithms

The value of α directly affects the number of outliers in the distribution of old data, which in turn influences the OR and the detection performance of this module. We will explore the relationship between the number of outliers, OR, and α through a mathematical derivation process.

Firstly, the data distribution of the ith cluster is assumed to be as follows:

$$D_i(x) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right), \tag{6}$$

where μ_i and σ_i represent the mean and standard deviation of the distribution D_i, respectively.

Secondly, we assume that when $x > N_i (N_i > \mu_i)$, all samples are outliers. The number of outliers can be estimated through the $D_i(x)$:

$$Outliers = \sum_{i=1}^{|C|} 2 \cdot d_i \cdot \int_{N_i}^{\infty} D_i(x)dx, \tag{7}$$

where $|C|$ represents the number of the clusters.

Let $t = \frac{x-\mu_i}{\sigma_i}$, then t follows the standard normal distribution $N(0,1)$.

$$\begin{aligned} \int_{N_i}^{\infty} D_i(x)dx &= \int_{\frac{N_i-\mu_i}{\sigma_i}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) dt \\ &= 1 - \Phi(\frac{N_i-\mu_i}{\sigma_i}), \end{aligned} \tag{8}$$

$$\Phi(\frac{N_i-\mu_i}{\sigma_i}) = \int_{-\infty}^{\frac{N_i-\mu_i}{\sigma_i}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) dt. \tag{9}$$

According to the Plackett's approximation and the Gaussian error function, it can be expressed as:

$$\Phi(\frac{N_i-\mu_i}{\sigma_i}) \approx \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{N_i-\mu_i}{\sqrt{2}\sigma_i}\right)\right], \tag{10}$$

$$\mathrm{erf}(\frac{N_i-\mu_i}{\sigma_i}) = \frac{2}{\sqrt{\pi}} \int_0^{\frac{N_i-\mu_i}{\sigma_i}} e^{-t^2}\, dt, \tag{11}$$

where $\mathrm{erf}(x)$ represents the Gaussian error function.

The $\mathrm{erf}(x)$ can be approximately represented using a series expansion as:

$$\mathrm{erf}(x) = 1 - \frac{e^{-x^2}}{x\sqrt{\pi}} \left[1 - \frac{1}{2x^2} + \frac{3}{4x^4} + O\left(\frac{1}{x^6}\right)\right]. \tag{12}$$

Let $N_i = C_i + CI$, C_i be the center of the ith cluster. According to Eq. (4), $\frac{N_i-\mu_i}{\sigma_i} = \frac{C_i+CI-\mu_i}{\sigma_i} = A\alpha + B$, A and B are constants. Consequently, combining Eq. (6) - Eq. (12), the number of outliers can be represented as:

$$Outliers(\alpha) \approx \sum_{i=1}^{C} d_i \cdot \frac{e^{-(A\alpha+B)^2}}{(A\alpha+B)\sqrt{\pi}}. \tag{13}$$

We discussed the relationship between $Outliers$ and α above, and the relationship between OR and α is also clear. To simplify the analysis, we assume that there exist a new data distribution D_j outside the confidence interval of D_i. Therefore, as α increases from 0, the confidence interval gradually expands, causing D_i and D_j to transition from a nonoverlapping state to overlapping, and eventually to being fully contained.

- Considering the nonoverlapping state, all samples in D_j are outliers, OR_1 can be represented as:

$$OR_1(\alpha) = \frac{\frac{1}{d'} \cdot d'}{\frac{1}{d} \cdot Outliers(\alpha)} = \frac{d}{Outliers(\alpha)}. \tag{14}$$

- Considering the overlapping or containing state, D_i have almost no outliers, which we set to 1. Meanwhile, the number of outliers in D_j will also gradually decrease. Similarly, its decreasing trend should be similar to $Outliers(\alpha)$, which we denote as $IDs(\alpha)$. Thus, OR_2 can be expressed as:

$$OR_2(\alpha) = \frac{\frac{1}{d'} \cdot IDs(\alpha)}{\frac{1}{d} \cdot 1} = \frac{d}{d'} \cdot IDs(\alpha). \tag{15}$$

To better illustrate the relationship between $Outliers$, OR, and α, we performed a graphical simulation of Eq. (13) - Eq. (15), the results are shown in Fig. 8.

Based on the graph, we can clearly observe that with an increase in α, $Outliers$ shows an exponential decay and OR initially increases with the growth of α, but as α increases excessively, OR also experiences an exponential decay. However, we note that the value near the inflection point of $Outliers$ may represent an optimal choice for α. At this point, OR can effectively detect data drift.

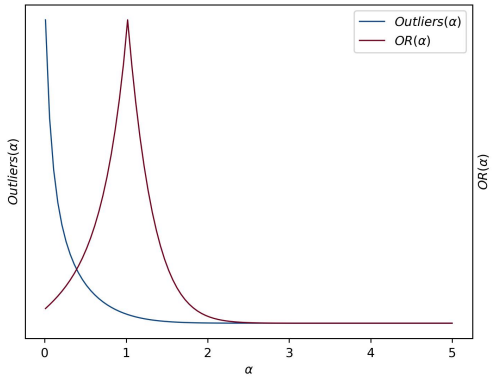


Fig. 8. Simulation of $Outliers(\alpha)$ and $OR(\alpha)$.

VII. Experimental Evaluation

A. Experiment Setup

Implementation. Our A-NIDS prototype is implemented in Python, using various libraries and frameworks. The *Detection Module* is built using *PyTorch*, the *Adaptive Module* leverages *Scikit-learn*, and the *Generation Module* is based on the original CTGAN code with modifications. Furthermore, we used libraries such as *pandas*, *numpy*, and *matplotlib*.

Tested. We deployed A-NIDS on a laptop with the following main configuration: a 13th Gen Intel® Core™ i5-13500 CPU, Windows 11, 16GB DDR5 5200Hz RAM, and 1TB SSD. Both the *Detection Module* and the *Adaptive Module* were trained and tested entirely on the CPU, without any involvement of the GPU.

Datasets. We selected two publicly available and widely used datasets, CICIDS-2017 [27], [28] and CSE-CICIDS-2018 [28], [29], to conduct our experiments. Both datasets use the CICFlowMeter feature extraction tool to extract around 80 flow-related features from the raw data packets, including length-related, time-related, and protocol-related features. We removed a few features, such as IP and Port, and ensured that the selected features remain consistent across both datasets used in the experiments (detailed feature usage can be found in **Section A of Appendix-I**).

CICIDS-2017 is used to simulate old data, and CSE-CICIDS-2018 is used to simulate new data. Both datasets contain *Benign* traffic and over ten types of attack traffic, such as Bruteforce (*FTP-Patator*, *SSH-Patator*), DoS attacks (*Hulk*, *GoldenEye*, *Slowloris*, *Slowhttptest*), Web attacks (*XSS* and *Brute-force*), *Infiltration* attacks, and *DDoS*, among others. We performed the following operations on the datasets to prepare them for the experiments:

- We applied random sampling to reduce training time and workload in our experiments;
- We removed features with attribute values equal to zero;
- We excluded attack traffic with very few samples, such as *XSS* and *Infiltration*;
- We split the datasets into training and testing sets in a 7:3 ratio.

Baselines. We used three ablation methods as baselines for comparison, in order to evaluate A-NIDS's ability to learn new knowledge and reduce catastrophic forgetting, demonstrating its superiority over fine-tuning methods.

- ***Mlp-2017***. A fully connected shallow neural network model trained on the CICIDS-2017 dataset.

TABLE I
DETECTION PERFORMANCE OF A-NIDS AND BASELINES ON CSE-CICIDS-2018 DATASET

Labels	Mlp-2017			Mlp-2018			Fine-Tuning			**A-NIDS**		
	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
Benign	0.3376	0.9631	0.5000	0.9976	0.9880	0.9928	0.9715	0.9706	0.9711	**0.9847**	**0.9904**	**0.9875**
FTP-Patator	0.3691	0.1038	0.1621	0.9191	0.3996	0.5570	0.5772	0.8633	0.6918	**0.6062**	**0.8747**	**0.7161**
SSH-Patator	0.9961	0.5214	0.6845	0.9999	0.9999	0.9999	0.9989	0.9975	0.9982	**0.9996**	**0.9999**	**0.9998**
DoS-GoldenEye	0.9997	0.3519	0.5206	0.9999	0.9999	0.9999	0.9687	0.9968	0.9825	**0.9997**	**0.9994**	**0.9995**
DoS-Hulk	0.9981	0.9465	0.9716	0.9980	0.9999	0.9990	0.9858	0.9694	0.9775	**0.9989**	**0.9997**	**0.9993**
Dos-Slowloris	0.9296	0.9413	0.9354	0.9966	0.9997	0.9981	0.9988	0.9805	0.9896	**0.9963**	**0.9999**	**0.9982**
Dos-SlowHttpTest	0.0000	0.0000	0.0000	0.6513	0.9696	0.7792	0.7780	0.4488	0.5692	**0.8215**	**0.5042**	**0.6249**
Bot	0.0000	0.0000	0.0000	0.9630	0.9985	0.9804	0.8641	0.9439	0.9022	**0.9745**	0.9407	**0.9573**
BruteForce	0.0000	0.0000	0.0000	0.9363	0.8909	0.9130	0.9590	0.6429	0.7697	0.9200	**0.8415**	**0.8790**
Weighted Avg	0.6309	0.5399	0.5240	0.9364	0.9173	0.9094	0.8988	0.8855	0.8818	**0.9209**	**0.9091**	**0.9065**

TABLE II
DETECTION PERFORMANCE OF A-NIDS AND BASELINES ON CICIDS-2017 DATASET

Labels	Mlp-2017			Mlp-2018			Fine-Tuning			**A-NIDS**		
	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
Benign	0.9891	0.9456	0.9668	0.4054	0.8109	0.5405	0.7003	0.8836	0.7813	**0.8111**	0.8131	0.8121
FTP-Patator	0.9970	0.9970	0.9970	0.0000	0.0000	0.0000	0.9917	0.6465	0.7827	0.8028	**0.9840**	**0.8842**
SSH-Patator	0.9510	0.9833	0.9669	0.6665	0.4908	0.5653	0.9422	0.8927	0.9168	0.9035	**0.9888**	**0.9442**
DoS-GoldenEye	0.9940	0.9981	0.9960	0.8587	0.2309	0.3640	0.9240	0.5619	0.6988	0.8913	**0.9710**	**0.9295**
DoS-Hulk	0.9646	0.9976	0.9808	0.6502	0.0800	0.1425	0.7420	0.9906	0.8484	**0.8761**	0.6918	0.7731
Dos-Slowloris	0.9971	0.9874	0.9922	0.2796	0.9960	0.4366	0.3943	0.9579	0.5586	**0.7502**	0.9365	**0.8331**
DoS-SlowHttpTest	0.9887	0.9964	0.9925	0.0000	0.0000	0.0000	0.6689	0.1069	0.1844	**0.9577**	**0.6094**	**0.7448**
Bot	0.8783	0.9740	0.9237	0.0000	0.0000	0.0000	0.4854	0.1360	0.2125	0.4159	**0.1467**	0.2169
BruteForce	0.9237	0.9348	0.9509	0.0004	0.0007	0.0005	0.3500	0.0557	0.0962	**0.8669**	**0.8514**	**0.8591**
Weighted Avg	0.9810	0.9804	0.9805	0.4106	0.3877	0.3059	0.7572	0.7020	0.6770	**0.8362**	**0.8385**	**0.8278**

- ***Mlp-2018***. The network structure is consistent with Mlp-2017, but the training dataset is CSE-CICIDS-2018.
- ***Fine-Tuning***. Based on Mlp-2017, the model was fine-tuned using the CSE-CICIDS-2018 dataset with 15 epochs and a learning rate of $1e-5$ (the learning rate for Mlp-2017 and Mlp-2018 is $1e-4$).

Metrics. We used five metrics to evaluate the performance of A-NIDS, including three commonly used metrics in machine learning algorithms: ***Precision***, ***Recall***, and ***F score***. Furthermore, we included two metrics that are crucial for practical deployment: ***FPR*** and ***FNR***.

B. Evaluation of Detection Performance

Tables I and Table II summarize the detection performance metrics (precision, recall, and F1 score) of A-NIDS and compare them with the baselines on the CSE-CICIDS-2018 and CICIDS-2017 datasets.

The Learning Ability to New Knowledge. As shown in Table I, we first observe that the *Mlp-2017* exhibits insufficient generalization when applied to the CSE-CICIDS-2018 dataset, showing low recall rates across multiple attacks such as FTP-Patator, DoS-GoldenEye, DoS-SlowHttpTest, Bot, and BruteForce. Meanwhile, it performs poorly in precision and F1 score. Secondly, we note that the *Fine-Tuning* achieves good detection performance on the CSE-CICIDS-2018 dataset, with multiple metrics maintained above 85%. Finally, we compare our *A-NIDS* with three baselines. The metrics indicate that our approach significantly outperforms the *Mlp-2017* and has a slight advantage over the *Fine-Tuning*, with performance comparable to the *Mlp-2018*. In general, the experimental results demonstrate that our *A-NIDS* exhibits superior detection performance when faced with new knowledge.

The Memory Ability of Old Knowledge. As shown in Table II, it is observed that the detection performance of *Mlp-2018* on the CICIDS-2017 dataset does not generalize effectively. This result is not surprising, as it does not incorporate learning from old knowledge. Secondly, we turn our attention to the *Fine-Tuning*. Although it performs well when facing new knowledge, its memory capacity is very poor when dealing with old knowledge. It experiences catastrophic forgetting on data from multiple categories, such as DoS-GoldenEye, DoS-SlowHttpTest, Bot, BruteForce, and others, resulting in overall poor performance. Finally, observing the detection performance of our *A-NIDS*, compared to the *Mlp-2018* and *Fine-Tuning*, we demonstrate a good memory ability for old knowledge. Except for the Bot attack data, we did not observe catastrophic forgetting. Considering the scarcity of Bot attack samples, training its generator posed difficulties, resulting in this phenomenon. Compared to *Mlp-2017*, we are slightly inferior in the detection performance of old knowledge, but this difference should be considered acceptable. Overall, while ensuring superior learning ability for new knowledge, we also demonstrate quite good memory ability for old knowledge.

In addition to the analysis of common evaluation metrics for machine learning algorithms mentioned above, we also evaluated two important metrics, FPR and FNR, which are crucial for the practical deployment of NIDS. The results are shown in Table III; A-NIDS demonstrates outstanding performance in

both datasets. Although the FPR in the CICIDS-2017 data set is 18.68%, which is higher compared to Mlp-2017 and Fine-Tuning, its FNR is only 6.01%, significantly lower than Mlp-2018 and Fine-Tuning. Considering that the old data involved in the A-NIDS training is generated, achieving this level of performance is excellent, and some degree of forgetting old knowledge is acceptable. Furthermore, in the CSE-CICIDS-2018 dataset, A-NIDS has an FPR of 0. 96%, the lowest among all methods, and an FNR of 0.26%, which is relatively low, only slightly higher than Mlp-2018 (0. 02%), with minimal difference. This indicates that A-NIDS not only significantly reduces the FNR but also greatly reduces it, outperforming other comparison methods. These results demonstrate that A-NIDS, while maintaining a low FPR, more effectively detects and prevents attacks, making it an exceptionally high performing NIDS.

TABLE III
FPR & FNR OF A-NIDS AND BASELINES ON TWO DATASETS

Methods	CICIDS-2017		CSE-CICIDS-2018	
	FPR	FNR	FPR	FNR
Mlp-2017	5.21%	0.33%	3.64%	32.71%
Mlp-2018	18.91%	37.75%	1.00%	0.02%
Fine-Tuning	11.64%	12.00%	2.52%	0.50%
A-NIDS	18.68%	**6.01%**	**0.96%**	**0.26%**

C. Evaluation of Adaptive Module

The Relationship between Outliers, OR and K. We conducted experiments with K values in the range $[9, 100]$. In each experiment, we recorded $Outliers$ in the old data distribution and OR. The relationships between $Outliers$, OR, and K are depicted using coordinate graphs, as shown in Fig. 9 (with the same α used in these experiments).

Firstly, as K increases, $Outliers$ gradually decreases, with the decrease rate slowing until it stabilizes near a constant value. The graph of $Outliers$ indicates that selecting a K value greater than the number of true labels (which is 9 in our data) is more beneficial to reduce $Outliers$, suggesting better convergence of the clustering model. However, an excessively large K is not optimal and may increase the risk of overfitting. Secondly, as K increases, OR also gradually increases, helping us to better identify occurrences of data drift. Similarly, an excessively large K does not yield better results. In Fig. 9, selecting a K value in the range of $[40, 50]$ appears to be optimal, rather than the number of true labels (1 benign and 8 attacks). This finding is consistent with the theoretical analysis presented in **Section VI - A**.

The Relationship between Outliers, OR and α. We conducted 100 experiments with α values in the range $(0, 10]$. For each experiment, we recorded $Outliers$ in the old data distribution and OR. The relationships among $Outliers$, OR, and α are illustrated using coordinate graphs, as shown in Fig. 10 (with the same K value used in these experiments).

We observe that $Outliers$ decreases exponentially with increasing α until it approaches 0. For OR, it initially increases with α, but when $Outliers$ reaches its inflection point (around

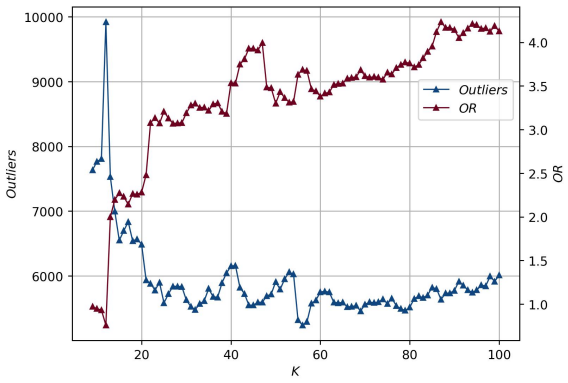


Fig. 9. The relationship between $Outliers, OR$ and K.

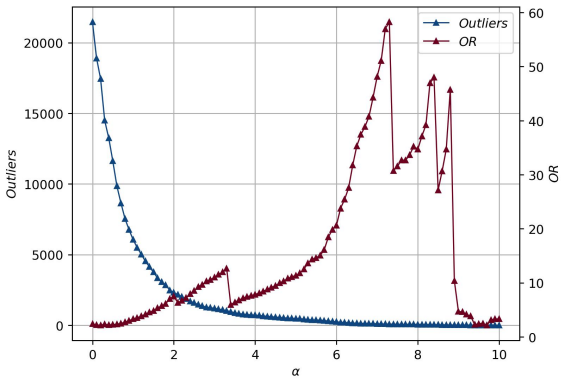


Fig. 10. The relationship between $Outliers, OR$ and α.

$\alpha = 7.0$ in Fig. 10), OR transitions from an increasing trend to a decreasing trend. This indicates that when α is close to 7, OR reaches its maximum, optimally reflecting the appearance of data drift. This observation is consistent with our theoretical analysis in **Section VI - B**. In practical deployment, we can determine the optimal α through experiments with old data, identifying the inflection point of the graph.

D. Evaluation of Generation Module

We used density graphs to visually analyze the differences between the real data and the generated data. The x-axis represents the feature values, while the y-axis represents the density. Due to space constraints, we randomly selected 10 features from the all features for visualization. We conducted two sets of experiments as described below.

StackedCTGAN & Real. We randomly selected data from one attack for analysis, and the results are shown in Fig. 11. It is evident that the density distributions of most features in the generated data closely resemble those of the real data, indicating that *StackedCTGAN* successfully captures the distribution characteristics of most features. The density distributions of these features exhibit multiple peaks, and the generated data align closely with the real data in terms of these multimodal distributions. This consistency demonstrates that

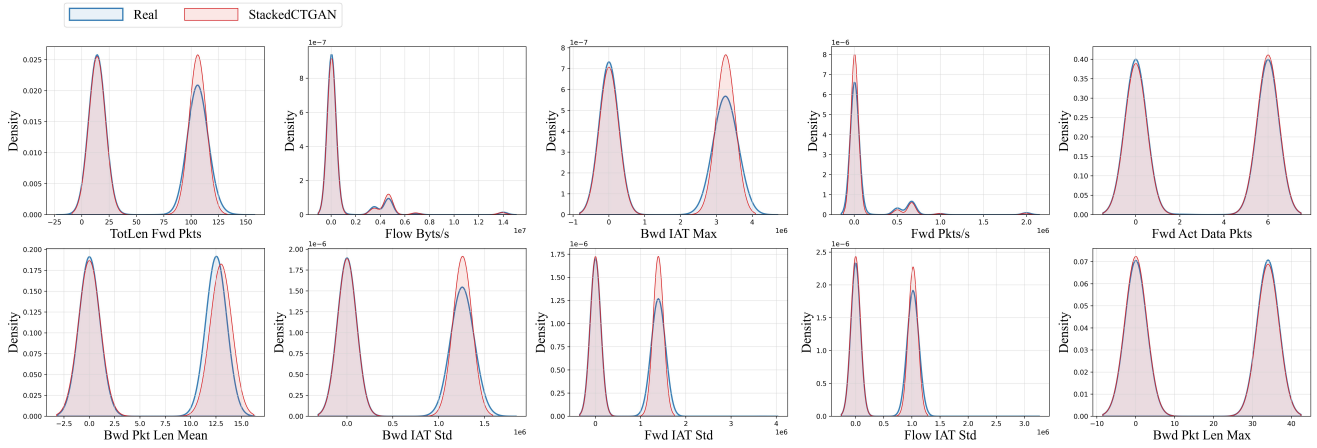
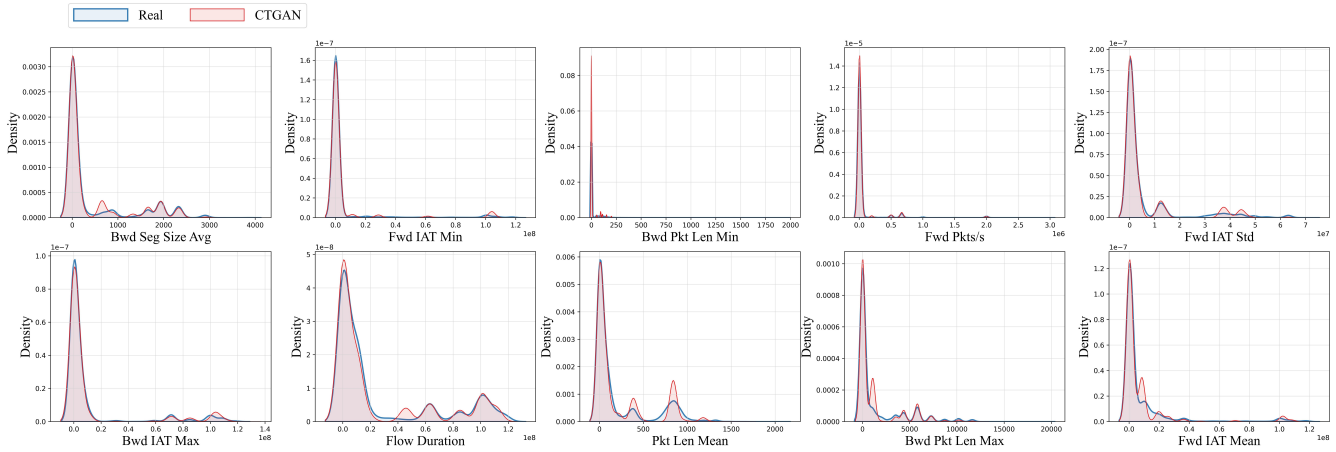
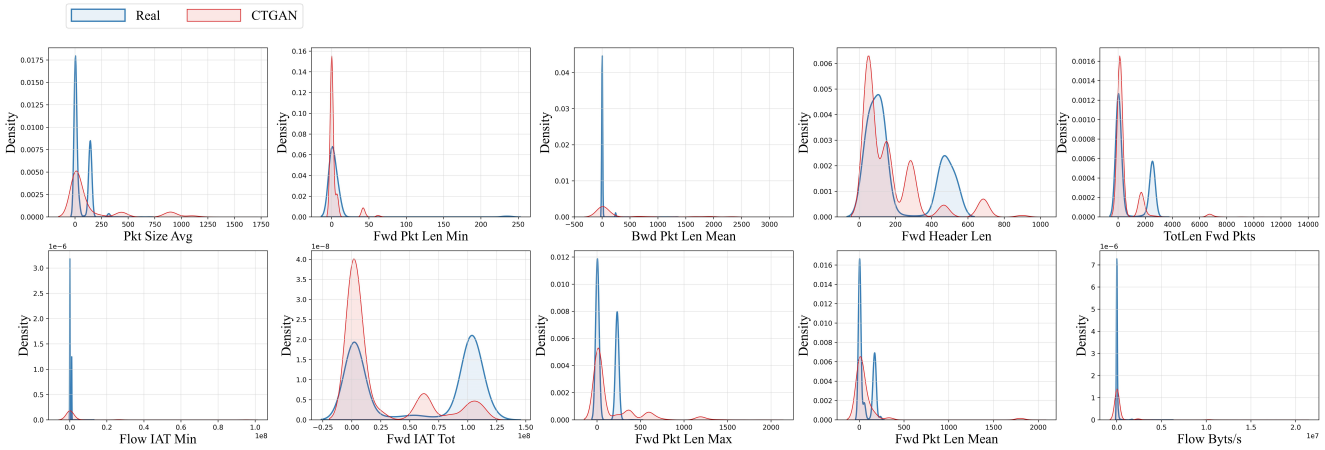


Fig. 11. Density graphs of generated data by StackedCTGAN and real data.



(a) Overall distributions.



(b) Local distributions.

Fig. 12. Density graphs of generated data by CTGAN and real data.

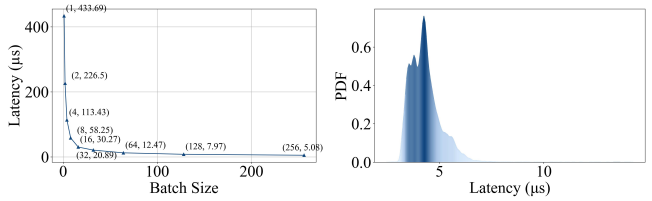
StackedCTGAN has strong capability in simulating complex distribution patterns.

CTGAN & Real. We also performed two sets of analyses on the data generated by *CTGAN*. We plotted the density graphs of the overall data and the data from randomly selected one attack. The results are shown in Fig. 12(a), we can note that the data generated by *CTGAN* closely resembles the overall distribution of the real data, demonstrating its strong generative capabilities. However, as shown in Fig. 12(b), we find discrepancies in the distribution compared to the real data. The number of peaks is inconsistent, indicating that *CTGAN* has some difficulties in capturing the inter-feature relationships accurately.

In contrast, our *StackedCTGAN* not only inherits *CTGAN*'s strong capability for generating overall data distributions but also effectively captures inter-feature relationships. Specifically, *StackedCTGAN* accurately simulates the complex distributions of various features, including long-tailed and multimodal distributions. This capability is validated by the density graphs presented above.

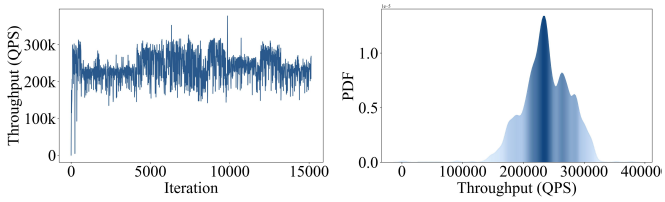
E. Performance Results

Model Size. Our detection model is a shallow, fully connected neural network with a total of 12,281 parameters, including both weights and biases. This shallow neural network achieves efficient classification performance while maintaining a low number of parameters, demonstrating its effectiveness and applicability in real-world applications.



(a) Latency vs. batch size. (b) Distribution.

Fig. 13. Detection latency.



(a) Throughput vs. iteration. (b) Distribution.

Fig. 14. Throughput.

Latency. We conducted tests for the detection latency of the *Detection Module* using 3,874,500 data samples and performed 9 experiments with varying batch sizes. Batch sizes ranged from 1, doubling with each increment. The experimental results are shown in Fig. 13(a), it can be observed that the overall trend of the curve is decreasing. The maximum detection latency is observed at batch size 1, at around 400 microseconds. When the batch size exceeds 64, the detection latency reduces to approximately 5 to 10 microseconds and stabilizes. These experiments demonstrate that the detection latency of our module is on the order of microseconds, with a maximum of just over 400 microseconds, indicating excellent real-time performance and making it suitable for practical deployment. Meanwhile, Fig. 13(b) illustrates the density distribution of detection latency for all samples with a batch size of 256. The density distribution indicates that the detection latency is mainly clustered around 5 microseconds. This experimental result further corroborates our findings of low detection latency, highlighting the suitability of our approach for real-time applications.

Throughput. We conducted throughput testing on the system with a batch size of 256, treating the testing process for each batch as an iteration. The throughput variations across all iterations, recorded for a total of 3,874,500 data samples, are shown in Fig. 14(a). The data indicate that the throughput fluctuates between 150,000 Queries Per Second (QPS) and 350,000 QPS, with the majority concentrated around 250,000 QPS. Meanwhile, Fig. 14 (b) shows the density distribution of the throughput, which aligns with the trends observed in Fig. 14 (a), further confirming that the throughput is predominantly centered around 250,000 QPS. In combination with the previous latency analysis, it is evident that our prototype system demonstrates excellent real-time performance, attributable to the shallow model and the limited number of model parameters employed. Moreover, considering that this performance was achieved on a low-specification platform using a Python-based prototype system, the results are quite satisfactory.

VIII. Discussion

Although machine learning provides superior capabilities for addressing complex problems in NIDS compared to traditional methods, it also encounters challenges common to all machine learning models, such as adversarial attacks. This section further discusses two prevalent types of adversarial attacks: poisoning and evasion [42].

Poisoning attacks occur when attackers intentionally inject malicious data into the training set during model training, to influence the model's learning process, thereby degrading its performance or causing erroneous decisions during testing or real-world applications [43]. Dataset purification is a widely used defense method against poisoning attacks. This approach is independent of the detection model, enabling data purification techniques to be directly integrated into NIDS.

Evasion attacks occur when attackers input carefully crafted malicious samples, often slightly modified normal samples, during the model testing phase to deceive the model into making incorrect predictions or classifications. Unlike poisoning attacks, evasion attacks take place after the model has been trained and deployed; The attackers do not alter the model itself or the training data, but instead manipulate the input data to achieve objectives [44]. Using adversarial examples and applying robustness optimization techniques, such as gradient clipping and regularization, are effective methods to defend

against evasion attacks, as these approaches can enhance the resilience of the model to attacks and reduce its sensitivity to adversarial samples.

The above discussion outlines common types of adversarial attacks on machine learning models, along with the corresponding defense strategies. This is crucial for the robustness of current machine learning-based NIDS, which will be the focus of our future work.

IX. Limitations and Future Work

In this section, we discuss some of the limitations of our A-NIDS and outline future work.

Limitations. Our proposed A-NIDS can adaptively detect data drift and utilize a generation model to produce old data, preventing catastrophic forgetting in the new model. However, this method has some limitations. Firstly, while we can identify data drift in unlabeled new data through unsupervised learning, training the new model still requires labeled new data, necessitating the involvement of domain experts for labeling. Nevertheless, our clustering and classification models can provide some guidance in this process. Secondly, A-NIDS faces the same issue as many deep learning methods: It lacks interpretability. The inability to provide clear explanations for detection results may lead to a lack of trust from non-expert users. Finally, obtaining a comprehensive, up-to-date, and realistic benchmark dataset for NIDS is hindered by privacy and proprietary concerns [45]. The substantial volume and sensitive characteristics of network traffic render manual labeling impractical. As a result, researchers often produce small, context-specific datasets, which typically do not capture the diversity of network traffic scenarios [46], [47]. The lack of a reliable, comprehensive and publicly accessible benchmark dataset represents a major challenge in the creation of an integrated IDS solution [45], [48]–[50].

Future Work. NIDS still faces many research challenges that require further investigation. For example, the issue of model confidence analysis – how to reasonably interpret our detection results or provide a confidence level for the detection results and explain their underlying basis. The detection performance of zero-day attacks remains a significant area for exploration. Particularly when considering the FPR, the current accuracy of zero-day intrusion detection remains suboptimal. Furthermore, collecting a dataset from a real-world network environment holds significant importance for NIDS research, particularly when considering complex network topologies, diverse attack types, and emerging threats. Addressing these challenges will be the focus of our future work.

X. Related Work

NIDS has been extensively researched for a considerable period. Existing methods primarily focus on traditional approaches based on signatures or handcrafted features [7]–[9], as well as novel techniques leveraging machine learning algorithms [18], [21], [22], [26], [51]–[55].

Traditional methods rely on predefined rule sets or models, which generally offer higher interpretability and lower complexity compared to machine learning-based technologies. For example, Wang et al. [7] proposed a method based on Symbolic Model Comparison, which extracts models from various popular implementations using symbolic execution in an offline phase. At runtime, it maintains an uncertain finite state machine to track the state of each possible implementation. In [8], Wu et al. introduced a distributed source-based intrusion detection method that prunes and extracts feature vectors from source dependencies at the system log level, stores them in an in-memory database, and detects results from multiple detectors through negotiation. In [9], Seo et al. introduced a streaming algorithm that uses sets of frequently occurring signatures identified from data streams for detection, achieving improved detection accuracy while using a small amount of fixed memory and a constant number of hash operations.

However, traditional NIDS methods are typically designed based on the characteristics or behaviors of known attacks, making them ineffective in detecting novel, unknown attacks and having limited detection capabilities for emerging threats such as zero-day attacks. Moreover, traditional NIDS require continuous updates or the addition of new rules to accommodate evolving attack patterns, which significantly increases management and maintenance costs.

With the rapid development of artificial intelligence in various fields, its application in NIDS has also been widely explored. For example, Yang et al. [26] proposed a two-stage minimization framework to decompose the intrusion detection problem, where the first stage seeks a scoring metric to minimize the empirical risk of misclassifying known attacks, and the second stage finds another scoring measure to minimize the inference risk of recognizing unknown attacks. Liu et al. [51] proposed a method with a two-layer detection module, one based on temporal and sequential logic, and the other based on historical data. For the first module, they differentiated between periodic and non-periodic messages based on changes in message intervals and used the idea of Markov Decision Processes (MDP) from reinforcement learning (RL) to automatically learn the logical relationships between sequences. The second module employed an online Sequential Extreme Learning Machine (OS-ELM) method to fit the data, further combining it with the Weibull distribution function for prediction and detection. In [52], Hang et al. proposed a pre-trained model that utilizes Masked Autoencoders (MAE) from the computer vision field for malicious network traffic classification. It adapts to a wide range of traffic lengths by utilizing burst (a generic representation of network traffic) and patch embedding, overcoming problems such as limited input length and fixed Byte Pair Encoding.

Furthermore, to mitigate the issue of false positives, Fu et al. [53] proposed an unsupervised method for identifying false positives in existing ML-based traffic detection systems without requiring prior knowledge of alerts. They considered traffic feature vectors associated with alerts as points in the traffic feature space and used point cloud analysis to capture the topological features between points for alert classification. To address the problem of data imbalance, Dina et al. [22] proposed a three-stage method. The first stage used self-supervised learning to learn latent patterns and robust representations from unlabeled data. The second

TABLE IV
COMPARISON OF RECENT INTRUSION DETECTION TECHNIQUES.

Methods	Used Technology	Interpretability	Low FPR	Zero-day Detection	Real-time	Data drift
Traditional IDS [7]–[9]	predefined rule sets	✓	×	×	✓	×
CVAE-EVT [26]	reconstruction-based	×	✓	✓	×	×
TLP-IDS [51]	MDP, RL and OS-ELM	×	×	×	✓	×
FLOW-MAE [52]	pre-trained model and MAE	✓	✓	×	×	×
pVoxel [53]	ML-Based	×	✓	×	✓	×
FS3 [22]	Contrastive Learning, KNN	×	×	×	×	×
FNN-Focal & CNN-Focal [25]	FNN, CNN, Focal loss	×	×	×	×	×
TMG-GAN [21]	GAN	×	✓	×	×	×
Kitsune [18]	ensemble of autoencoders	×	✓	✓	✓	×
Our A-NIDS	Mlp, K-Means and CTGAN	×	✓	✓	✓	✓

stage introduced few-shot learning with contrastive training to eliminate the dependency on a large amount of labeled data. The third stage introduced a K-nearest neighbor algorithm that subsampled majority class instances to further reduce imbalance and enhance overall performance. Dina et al. [25] proposed using a loss function called focal loss to address the data imbalance issue in IoT intrusion detection. This method automatically reduces the weight of easily classified samples, focusing on difficult-to-classify negative samples. By dynamically scaling the gradient updates, this approach effectively trains machine learning models. In [21], Ding et al. proposed a data augmentation method based on GAN. For real-time applications, Mirsky et al. [18] proposed to learn to detect attacks on unsupervised local networks online. Their core algorithm (Kitsune) used an ensemble of autoencoders to jointly differentiate between normal and abnormal traffic patterns.

We summarize the most recent research work, highlighting their advantages and limitations, as presented in Table IV. Previous studies have addressed various issues in NIDS and have achieved significant progress; however, they almost universally operate under the assumption that data distributions are stationary. This assumption is nearly impossible in real-world applications. Although network data distributions can be assumed to be stationary over a short period of time, they will inevitably drift after a period of time, causing data with the same label to change. This phenomenon is unavoidable and indicates that our model has become outdated, necessitating an update to adapt to a new data environment. It is important to note that this is not entirely equivalent to a zero-day attack; rather, it signifies that our model needs to be updated to cope with the evolving data landscape.

Based on this analysis, this study aims to develop an adaptive NIDS to detect data drift and promptly update the model accordingly. Overall, our contributions lie in using a clustering model to simulate the data distribution to detect potential data drift. We also theoretically derive the outlier ratio threshold for detection. Furthermore, we address the issue of catastrophic forgetting by retraining the model with both new and old data generated via *StackedCTGAN*, which also solves the problem of continuous data storage accumulation. Notably, our detection model is lightweight and better suited for real-time requirements in intrusion detection.

XI. Conclusion

We propose an adaptive network intrusion detection system named A-NIDS, designed to address issues such as data drift, catastrophic forgetting, and real-time deployment in real-world. Unlike previous methods, we prioritize real-time requirements and decompose the complex issues in intrusion detection into a main task and two bypass tasks. The main task, named *Detection Module*, ensures good detection performance while maintaining low detection latency. To achieve this, we select a shallow, fully connected neural network structure. Bypass task 1, named *Adaptive Module*, consists of a clustering model, assisting the main task in detecting data drift in an unsupervised manner. We provide thorough theoretical analysis of this module. Bypass task 2 is a *Generation Module* capable of generating old data with similar distributions, addressing catastrophic forgetting during new model updates and the storage cost issue caused by accumulating old data. Finally, we conduct extensive experiments on two publicly available datasets, CICIDS-2017 and CSE-CICIDS-2018, demonstrating not only excellent detection performance for both new and old data but also low detection latency, as low as approximately 5 microseconds.

Acknowledgments

This work is supported by the Key Research and Development Program of Zhejiang Province (No. 2024SSYS0001 and No. 2023C01001).

Appendix

Appendix-I Details of Implementations

A. Explanation of The Features Used

The features of the dataset we used were extracted by CICFlowMeter [56]. All features used are listed in Table V to facilitate replication by future researchers. The explanations of the features can be found in [29].

B. Experimental Results of Detection Latency under Different Batch Sizes

Fig. 15 presents the results of 6 experiments conducted under different batch size conditions to measure detection latency. We conduct detailed measurements and statistics for various batch sizes, including 1, 2, 4, 8, 16, 32, 64, 128 and 256.

TABLE V
FEATURES USED IN OUR EXPERIMENTS.

Type	Features
Time-Realted	Flow Duration, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min, Fwd IAT Tot, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min, Bwd IAT Tot, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Max, Bwd IAT Min
Packet-Realted	Tot Fwd Pkts, Tot Bwd Pkts, Flow Byts/s, Flow Pkts/s, Fwd Pkts/s, Bwd Pkts/s, Subflow Fwd Pkts, Subflow Fwd Byts, Subflow Bwd Pkts, Subflow Bwd Byts, Init Fwd Win Byts, Init Bwd Win Byts, Fwd Act Data Pkts
Length-Realted	TotLen Fwd Pkts, TotLen Bwd Pkts, Fwd Pkt Len Max, Fwd Pkt Len Min, Fwd Pkt Len Mean, Fwd Pkt Len Std, Bwd Pkt Len Max, Bwd Pkt Len Min, Bwd Pkt Len Mean, Bwd Pkt Len Std, Fwd Header Len, Bwd Header Len, Pkt Len Min, Pkt Len Max, Pkt Len Mean, Pkt Len Std, Pkt Len Var, Pkt Size Avg, Fwd Seg Size Avg, Bwd Seg Size Avg, Fwd Seg Size Min
Protocol-Related	FIN Flag Cnt, SYN Flag Cnt, RST Flag Cnt, PSH Flag Cnt, ACK Flag Cnt, URG Flag Cnt, ECE Flag Cnt, Down/Up Ratio, Fwd PSH Flags

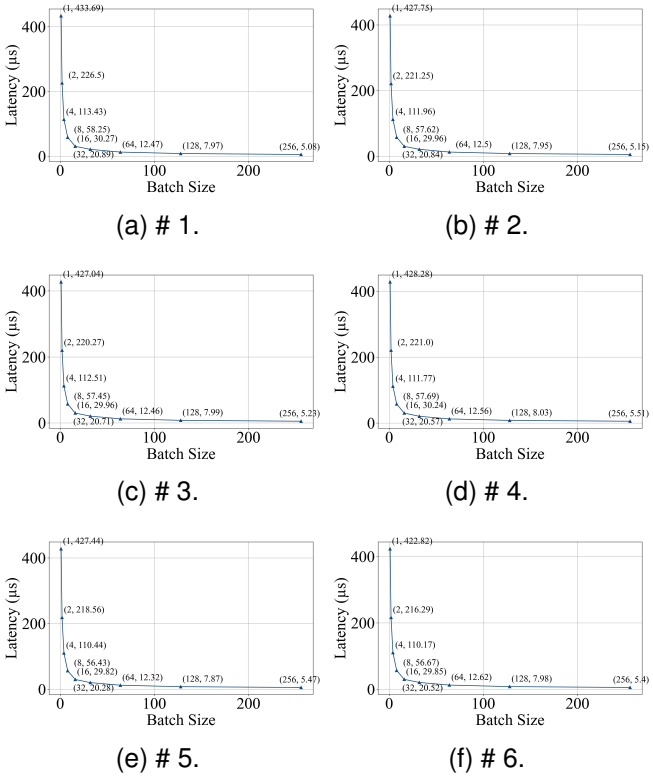


Fig. 15. Detection latency under different batch sizes.

REFERENCES

[1] Y. Zou, J. Zhu, X. Wang, and L. Hanzo, "A survey on wireless security: Technical challenges, recent advances, and future trends," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1727–1765, 2016.

[2] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating critical security issues of the iot world: Present and future challenges," *IEEE Internet of things journal*, vol. 5, no. 4, pp. 2483–2495, 2017.

[3] W. Li, W. Meng, and L. F. Kwok, "Surveying trust-based collaborative intrusion detection: state-of-the-art, challenges and future directions," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 280–305, 2021.

[4] R. Mitchell and R. Chen, "Behavior-rule based intrusion detection systems for safety critical smart grid applications," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1254–1263, 2013.

[5] V. Kumar, D. Sinha, A. K. Das, S. C. Pandey, and R. T. Goswami, "An integrated rule based intrusion detection system: analysis on unsw-nb15 data set and the real time online dataset," *Cluster Computing*, vol. 23, pp. 1397–1418, 2020.

[6] F. Erlacher and F. Dressler, "On high-speed flow-based intrusion detection using snort-compatible signatures," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 495–506, 2020.

[7] Z. Wang, S. Zhu, K. Man, P. Zhu, Y. Hao, Z. Qian, S. V. Krishnamurthy, T. La Porta, and M. J. De Lucia, "Themis: Ambiguity-aware network intrusion detection based on symbolic model comparison," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3384–3399.

[8] Y. Wu, Y. Xie, X. Liao, P. Zhou, D. Feng, L. Wu, X. Li, A. Wildani, and D. Long, "Paradise: real-time, generalized, and distributed provenance-based intrusion detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1624–1640, 2022.

[9] H. Seo and M. Yoon, "Generative intrusion detection and prevention on data stream," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4319–4335.

[10] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE communications surveys & tutorials*, vol. 21, no. 1, pp. 686–728, 2018.

[11] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3366–3383.

[12] C. Zha, Z. Wang, Y. Fan, X. Zhang, B. Bai, Y. Zhang, S. Shi, and R. Zhang, "Skt-ids: Unknown attack detection method based on sigmoid kernel transformation and encoder–decoder architecture," *Computers & Security*, vol. 146, p. 104056, 2024.

[13] B. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," in *2016 8th IEEE international conference on communication software and networks (ICCSN)*. IEEE, 2016, pp. 581–585.

[14] H. Yao, D. Fu, P. Zhang, M. Li, and Y. Liu, "Msml: A novel multilevel semi-supervised machine learning framework for intrusion detection system," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1949–1959, 2018.

[15] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE transactions on emerging topics in computational intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[16] L. Yang, A. Moubayed, and A. Shami, "Mth-ids: A multitiered hybrid intrusion detection system for internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616–632, 2021.

[17] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE access*, vol. 6, pp. 33 789–33 795, 2018.

[18] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[19] W. Wang, X. Du, D. Shan, R. Qin, and N. Wang, "Cloud intrusion detection method based on stacked contractive auto-encoder and support vector machine," *IEEE transactions on cloud computing*, vol. 10, no. 3, pp. 1634–1646, 2020.

[20] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection based on meta-learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3540–3552, 2020.

[21] H. Ding, Y. Sun, N. Huang, Z. Shen, and X. Cui, "Tmg-gan: Generative adversarial networks-based imbalanced learning for network intrusion

detection," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1156–1167, 2023.

[22] D. Ayesha S and S. AB, "Fs3: Few-shot and self-supervised framework for efficient intrusion detection in internet of things networks," in *Proceedings of the 39th Annual Computer Security Applications Conference*, 2023, pp. 138–149.

[23] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3431–3446.

[24] X. Wang, "Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 785–798.

[25] A. S. Dina, A. Siddique, and D. Manivannan, "A deep learning approach for intrusion detection in internet of things using focal loss function," *Internet of Things*, vol. 22, p. 100699, 2023.

[26] J. Yang, X. Chen, S. Chen, X. Jiang, and X. Tan, "Conditional variational auto-encoder and extreme value theory aided two-stage learning approach for intelligent fine-grained known/unknown intrusion detection," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3538–3553, 2021.

[27] C. I. for Cybersecurity, "Cicids 2017 dataset," 2017, https://www.unb.ca/cic/datasets/ids-2017.html.

[28] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[29] C. I. for Cybersecurity (CIC), "Cicids 2018 dataset," 2018, https://www.unb.ca/cic/datasets/ids-2018.html.

[30] H. V. Vo, H. P. Du, and H. N. Nguyen, "Apelid: Enhancing real-time intrusion detection with augmented WGAN and parallel ensemble learning," *Computers & Security*, vol. 136, p. 103567, 2024.

[31] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[32] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[34] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *International conference on machine learning*. PMLR, 2016, pp. 1558–1566.

[35] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in neural information processing systems*, vol. 32, 2019.

[36] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[37] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola, "Integrating structured biological data by kernel maximum mean discrepancy," *Bioinformatics*, vol. 22, no. 14, pp. e49–e57, 2006.

[38] D. I. Belov and R. D. Armstrong, "Distributions of the kullback–leibler divergence with applications," *British Journal of Mathematical and Statistical Psychology*, vol. 64, no. 2, pp. 291–309, 2011.

[39] A. Mosquera, "Tackling data drift with adversarial validation: an application for german text complexity estimation," in *Proceedings of the GermEval 2022 Workshop on Text Complexity Assessment of German Text*, 2022, pp. 39–44.

[40] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of data science*, vol. 2, pp. 165–193, 2015.

[41] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija, and J. Heming, "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023.

[42] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 321–338.

[43] M. Surekha, A. K. Sagar, and V. Khemchandani, "A comprehensive analysis of poisoning attack and defence strategies in machine learning techniques," in *2024 IEEE International Conference on Computing, Power and Communication Technologies (IC2PCT)*, vol. 5. IEEE, 2024, pp. 1662–1668.

[44] S. Wang, R. K. Ko, G. Bai, N. Dong, T. Choi, and Y. Zhang, "Evasion attack and defense on machine learning models in cyber-physical systems: A survey," *IEEE Communications Surveys & Tutorials*, 2023.

[45] S. Layeghy, M. Gallagher, and M. Portmann, "Benchmarking the benchmark—comparing synthetic and real-world network ids datasets," *Journal of Information Security and Applications*, vol. 80, p. 103689, 2024.

[46] R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, "Zero-day attack detection: a systematic literature review," *Artificial Intelligence Review*, vol. 56, no. 10, pp. 10 733–10 811, 2023.

[47] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 7–12.

[48] Z. Azam, M. M. Islam, and M. N. Huda, "Comparative analysis of intrusion detection systems and machine learning based model analysis through decision tree," *IEEE Access*, 2023.

[49] J. Hesford, D. Cheng, A. Wan, L. Huynh, S. Kim, H. Kim, and J. B. Hong, "Expectations versus reality: Evaluating intrusion detection systems in practice," *arXiv preprint arXiv:2403.17458*, 2024.

[50] A. Singh, J. Prakash, G. Kumar, P. K. Jain, and L. S. Ambati, "Intrusion detection system: A comparative study of machine learning-based ids," *Journal of Database Management (JDM)*, vol. 35, no. 1, pp. 1–25, 2024.

[51] X. Liu, D. He, Y. Gao, S. Zhu, and S. Chan, "Tlp-ids: A two-layer intrusion detection system for integrated electronic systems," in *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020, pp. 205–214.

[52] Z. Hang, Y. Lu, Y. Wang, and Y. Xie, "Flow-mae: Leveraging masked autoencoder for accurate, efficient and robust malicious traffic classification," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 297–314.

[53] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ml-based malicious traffic detection: Reducing majorities of false positive alarms," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1005–1019.

[54] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-gaussian: provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2658–2674, 2019.

[55] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.

[56] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *International Conference on Information Systems Security and Privacy*, vol. 2. SciTePress, 2017, pp. 253–262.