Detecting and Adapting to Stealthy Label-Inversion Drifts via Conditional Distribution Inference

Xiaoli Zhang
University of Science and Tech Beijing
xiaoli.z@ustb.edu.cn

Yue Xiao
Tsinghua University
yue-xiao-25@outlook.com

Qilei Yin*
Zhongguancun Laboratory
yinql@zgclab.edu.cn

Zhengyang Li
University of Science and Tech Beijing
U202343381@xs.ustb.edu.cn

Xinyan Wang
China Unicom Digital Tech Co., Ltd
wangxy5002@chinaunicom.cn

Jianrong Zhang
China Unicom Digital Tech Co., Ltd
zhangjr67@chinaunicom.cn

Ke Xu
Tsinghua University
xuke@tsinghua.edu.cn

Qi Li
Tsinghua University
qli01@tsinghua.edu.cn

Xu-Cheng Yin
University of Science and Tech Beijing
xuchengyin@ustb.edu.cn

***Abstract*—Deep learning (DL) based malicious traffic detectors have been widely developed to detect diverse network attacks, yet they are suffering from significant performance degradation due to concept drift. Existing anti-concept drift arts focus on combating the drifting traffic whose features significantly diverge from training traffic. However, they neglect a stealthy yet common situation where the testing traffic has similar features to the training traffic but with *opposite* ground truth labels. As a result, the DL-based detectors would always make incorrect predictions for the stealthy drifting traffic, insufficient to perform long-term real-world intrusion detection.**

In this paper, we propose `Chameleon`, a novel active learning framework that combats stealthy drifting traffic by inferring the conditional distribution of the testing traffic with small manual labeling overhead. Specifically, `Chameleon` measures the fine-grained correlations between the high-dimensional and heterogeneous testing traffic and selects a small number of highly representative testing traffic samples for manual labeling, to accurately infer other testing samples' labels. With the inferred labels, `Chameleon` checks the conditional distribution shift from the training to testing traffic to detect concept drift and incrementally trains the DL-based detectors to make them effectively adapt to the shifted distribution. Extensive experiments with six supervised and unsupervised DL-based detectors on three public and four synthetic datasets show that, under stealthy drifting traffic, `Chameleon` improves the AUT of the DL-based detectors by a range of 18.53% to 23.89%, while the improvement of SOTA baselines is only between 0.06% and 1.86%.

***Index Terms*—Malicious Traffic Detection, Concept Drifts, Label Inversion, Conditional Distribution Inference**

I. Introduction

Malicious traffic detection is one of the most critical tasks for cyber security [29], [11]. Recently, deep learning (DL) based methods [44], [17], [18], [68], [31] have achieved promising performance in detecting various malicious traffic, showing great potential for a new security paradigm. However, DL models heavily rely on an assumption that the training and testing data are identically and independently distributed [10], [60]. In real networks, newly incoming testing traffic is usually generated by diverse network activities, making its distribution diverge from that of the DL models' training traffic. This phenomenon, known as *concept drift* [41], [42], might lead to performance degradation of the DL models (i.e., more incorrect predictions), hindering them from being deployed for long-term intrusion detection.

The traditional anti-concept drift strategies [49], [67], [33], [43] collect all testing traffic over some time for manual investigation and labeling. Then, they retrain the DL models based on the labeled testing traffic. While manually labeling all testing traffic is quite expensive in practice, recent active learning methods [32], [8], [70], [12], [22] identify the significant testing samples that are likely to introduce distribution drift for manual labeling and further drift adaptation. Specifically, they learn the *marginal feature distribution from the training samples* and select the testing samples whose *features significantly deviate from the learned marginal distribution* for manual labeling. Finally, the DL models are incrementally trained based on the labeled testing samples to learn the shifted new distribution.

However, these methods overlook an important yet underexplored form of concept drift, referred to as *stealthy label-inversion drift*[1] in this paper, where testing traffic shares similar feature distributions with the training data but has opposite ground-truth labels. This subtle form of drift is especially challenging to discover because the features of drifting samples remain close to the marginal feature distribution of the training data, making prior methods that rely on marginal feature distribution divergence ineffective. More crucially, stealthy label-inversion drift is a real-world phenomenon driven by many common network events such as mimic attacks [16], [67], [65], [71] and adversarial traffic

*Corresponding author.

[1]We refer to traffic that is subject to stealthy label-inversion drift as *stealthy drifting traffic*.

generation [9], [73]. For instance, attackers manipulate botnets to generate malicious traffic that closely resembles normal web browsing patterns, exhibiting statistical feature similarities to legitimate traffic such as similar probabilities of requesting different webpages and comparable average number of page requests [71]. Further, we conducted a measurement study on the long-term Kyoto 2006+ dataset [61] to demonstrate the prevalence of stealthy drifting traffic in practice. Using cluster-based similarity and confidence-based misclassification filtering (see § II-A), we found that 15.49% to 40.74% of the mispredicted samples are likely to be stealthy drifting traffic. Thus, combating stealthy drifting traffic is essential for long-term real-world malicious traffic detection.

In this paper, we propose `Chameleon`, a novel active learning framework designed to effectively detect and adapt to stealthy drifting traffic. The core idea is to infer the conditional distribution of testing traffic and estimate its divergence from the training distribution. Since the conditional distribution captures the relationship between features and labels, stealthy label-inversion drift naturally leads to observable shifts, providing a reliable signal for detection. This approach also generalizes to traditional non-stealthy drift scenarios [32], [8], [70], [67], [22], [12], where both marginal and conditional distributions evolve, making `Chameleon` a universal solution for diverse concept drift types that undermine the performance of DL-based malicious traffic detectors. Further, our framework incrementally updates the DL-based detectors based on the inferred distribution, enabling adaptation to the shifted distribution and improving long-term detection performance.

Inferring the conditional distribution of testing traffic requires label information, which is typically annotated by security experts and is very costly. To reduce manual labeling overhead, `Chameleon` exploits the intrinsic correlations within the testing traffic, selecting a small number of highly representative samples for manual labeling, and propagating their labels to the remaining samples based on measured correlations. This design leverages the observation that network traffic *within the same period* is often generated by recurring or similar activities, leading to strong behavioral correlations [18], [52]. For example, encrypted flows of repetitive SSH cracking attempts have almost identical packet-level features [18]. Note that this observation is temporally localized and does not conflict with the concept drift encountered by DL models during long-term deployment.

In detail, `Chameleon` introduces three key modules: (i) An ensemble clustering based traffic correlation measurement module captures the fine-grained correlations between the high-dimensional and heterogeneous testing traffic. (ii) A novel label inference module formulates a multi-objective optimization function to select a small number of highly representative testing traffic data in a distribution-aware manner for manual labeling, and propagates their labels to the rest. (iii) A drift detection and adaptation module identifies the conditional distribution shifts based on the inferred labels for drift detection, and introduces a novel incremental training loss to adapt the detector to the new distribution while handling label noises (originating from label inference) and spatiotemporal imbalance. Once adapted, the DL-based detector achieves improved performance on subsequent drifting traffic. Overall, our contributions are:

- We present a novel active learning framework called `Chameleon` that combats stealthy label-inversion drifts by inferring the conditional distribution of the testing data in a model-agnostic and labor-efficient manner. This idea enables `Chameleon` to handle diverse drift types and supports both supervised and unsupervised DL models.
- `Chameleon` devises a novel label inference method that selects highly representative samples in a distribution-aware manner for manual labeling and accurately infers labels of the rest. `Chameleon` also designs a new incremental training loss function that can be applied to any DL-based malicious traffic detector to learn new distributions even with label noises and imbalanced data.
- We instantiate `Chameleon` with six supervised and unsupervised DL models and evaluate it against five SOTA baselines across three public and four synthetic datasets. Experimental results show that, under stealthy label-inversion drift, `Chameleon` improves the AUT of the DL-based detectors by a range of 18.53% to 23.89%, while the improvement of SOTA baselines is only between 0.06% and 1.86%. Moreover, our method only uses about 1% of the manual labeling cost of the baselines while achieving better detection performance.

II. Motivation and Problem Statement

A. Motivation

In the machine learning community, concept drift is commonly defined as a change in the joint probability distribution of input features and labels [19], denoted as $P(x, y) = P(x) \cdot P(y|x)$, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Unlike prior work in the security community [32], [8], [70], [22], [12] that handles the drift with both the marginal distribution $P(x)$ and the conditional distribution $P(y|x)$ changing simultaneously, this paper focuses on a special concept drift situation, called stealthy label-inversion drift, in which $P(x)$ remains stable while $P(y|x)$ changes.

In real-world networks, stealthy label-inversion drift refers to cases where testing traffic shares similar features with training traffic but has opposite labels. Such drift can be caused by many network events like mimic attacks [16], [67], [65], [71] and adversarial traffic generation [9], [73], and can lead to performance degradation of DL-based malicious traffic detectors after deployment. For example, Yu et al. [71] discover that normal web accessing behavior follows a Zipf-like distribution and control botnets to generate malicious traffic that mimics the measured normal distribution. The malicious traffic shares similar statistical features, e.g., the probabilities for requesting different web pages and the average number of web requests, with normal web browsing traffic. Abdullah et al. [9] generate adversarial traffic by mixing malicious packets with benign ones. The resulting traffic closely resembles normal traffic in

Table I: The average number and proportion of stealthy drifting traffic samples between 2014 and 2015 in the Kyoto dataset. Stealthy Normal (Malicious) are the normal (malicious) testing samples that have similar features to malicious (normal) training data and are misclassified as malicious (normal).

Window Len.	Stealthy Normal		Stealthy Malicious		Model Performance	
	#	Ratio in FP	#	Ratio in FN	F1 (Val.)	F1 (Test)
One Month	465.7	19.13%	456.1	15.49%	82.70%	74.99%
One Quarter	1792.9	27.73%	1762.6	17.20%	81.03%	73.67%
Half Year	5613.3	40.74%	3067.8	15.53%	82.98%	75.27%

key features, such as TCP flag patterns, thus easily evading the detection of NIDS.

To further verify the prevalence of stealthy drifting traffic in real-world networks, we conduct a measurement study using the long-term Kyoto 2006+ dataset [61]. We choose the traffic samples from 2014 to 2015 and divide them into consecutive time windows. The earliest window is randomly split into training and validation sets (4:1 ratio), while the remaining windows serve as the testing sets. We represent each sample using the original features provided by the dataset and adopt a cross-validation method to identify likely stealthy drifting samples. Specifically, we train a Multilayer Perceptron (MLP)-based malicious traffic detector [12] on the earliest time window. We also apply DBSCAN [55] to cluster normal and malicious training samples separately, and calculate the average intra-cluster distance of each cluster as its threshold. A testing sample is flagged as stealthy drifting with high confidence if two conditions satisfy simultaneously: (i) The detector misclassifies this sample with high prediction confidence (> 0.8), (ii) The distance from this sample to the center of a cluster whose label is opposite to this sample is smaller than the cluster's threshold. To avoid experimental bias, we use three window lengths, including one month, one quarter, and half a year, individually. For each window length, we evaluate the detector's performance on the validation and testing sets, identify the stealthy drifting samples per window, and report their average proportion to the incorrectly predicted testing samples, i.e., false positives (FP) and false negatives (FN).

As shown in Table I, the detector's performance on the testing sets is consistently lower than on the validation set, indicating the presence of concept drift. On average, each window contains hundreds to thousands of stealthy drifting samples, accounting for 15.49% to 40.74% of the misclassified testing samples. These findings suggest that stealthy label-inversion drift is a primary contributor to performance degradation in DL-based traffic detectors during long-term deployment. However, existing active learning approaches [49], [67], [33], [43] struggle to defend against stealthy label-inversion drift, as they primarily focus on detecting samples that deviate from the marginal feature distribution, while stealthy label-inversion drift manifests as a conditional distribution shift. This highlights the need for new techniques capable of detecting and adapting to stealthy label-inversion drift.

B. Problem Statement

This paper aims to develop a novel active learning framework that can combat stealthy label-inversion drift for preserving the detection performance of DL-based malicious traffic detectors with small manual labeling overhead. Formally, given a training traffic sample set D_{train} and N testing traffic sample sets $D^1_{\mathsf{test}}, ..., D^N_{\mathsf{test}}$, which can be formatted as:

$$D_{\mathsf{train}} = \{(X_i, y_i)\}_{i=1}^{\mathsf{train}},\ D^k_{\mathsf{test}} = \{(X_i, y_i)\}_{i=1}^{N_k},$$
$$where\ X_i \in \mathbb{R}^d,\ y_i \in Y = \{0, 1\}.$$

Note that $\{(X_i, y_i)\}$ denotes the pair of a network traffic sample X_i (e.g., a TCP flow or an HTTP request) and its label y_i, where 0 and 1 represent normal and malicious, respectively. A DL-based malicious traffic detector f is trained based on D_{train} at first, then it is deployed to predict each testing set D_{test} in their chronological order. Assume the stealthy label-inversion drift occurs starting from the k-th testing sets, i.e., some testing traffic samples belonging to the k-th and subsequent testing sets have similar features with the training samples yet with opposite ground-truth labels. As a result, the detector f often incorrectly predicts these testing traffic samples as the labels of training samples, significantly undermining the detection performance on the k-th and subsequent testing sets. This drift can be formatted as:

$$\exists (X_j, y_j) \in \cup\{D^m_{\mathsf{test}}\}_{m=k}^{N}, \exists (X_i, y_i) \in D_{\mathsf{train}},\ |X_i - X_j| < \epsilon,$$
$$s.t.\ \ y_j \neq y_i\ \wedge\ f_{\sim D_{\mathsf{train}}}(X_j) = f_{\sim D_{\mathsf{train}}}(X_i) = y_i$$

where ϵ is a small threshold that determines the similarity between traffic features.

Our objective is to detect stealthy drifting traffic in time with small manual labeling overhead, i.e., by labeling only a small subset $D_{\mathsf{label}} \subset D_{\mathsf{test}} = \cup\{D^k_{\mathsf{test}}\}_{k=1}^{N}$, where $|D_{\mathsf{label}}| \ll |D_{\mathsf{test}}|$, and make the detector f efficiently adapt to the drift, to improve its performance on future traffic. Note that, in existing anti-concept drift studies [32], [8], [70], [67], [22], [12], it is a common and necessary practice to query some ground-truth labels from security analysts to confirm whether drift has occurred and enable subsequent drift adaptation. Our framework follows this practice, but further aims to minimize the labeling overhead throughout the entire process. Besides, it should also be applicable to both supervised and unsupervised DL-based malicious traffic detectors, where supervised detectors [30], [68] are trained on both normal and malicious traffic while unsupervised ones [44], [15], [14] only use normal training traffic. The recent arts [40], [37] pre-trained on large-scale unlabeled datasets and fine-tuned on small amounts of labeled traffic traffic are also treated as supervised in our setting.

III. Methodology

A. Overview

`Chameleon` detects and adapts to stealthy drifting traffic by inferring the conditional distribution of testing traffic and identifying divergences from the training distribution. To achieve this, it mines the intrinsic correlations of the testing traffic, selects a small number of highly representative

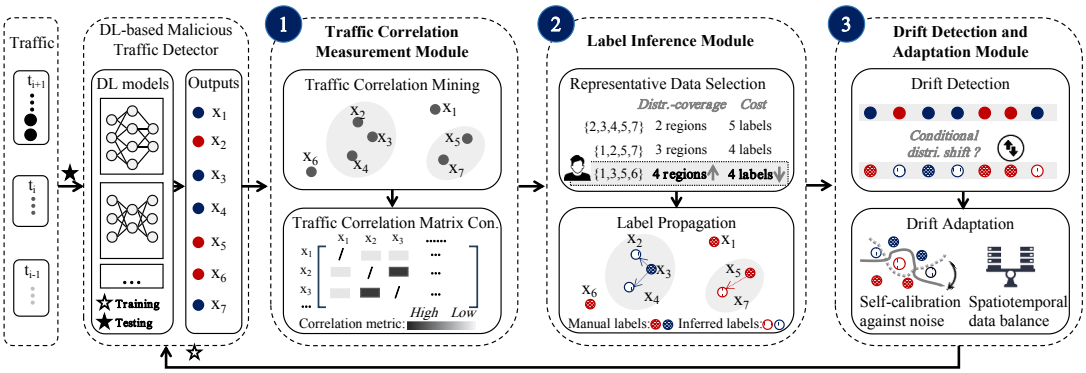


Figure 1: The overview of Chameleon.

testing traffic for manual labeling, and infers the labels of the rest based on the measured correlations. By comparing the inferred labels with the predictions of the DL-based detector, Chameleon detects conditional distribution shifts which indicate concept drift. The inferred labels are also used to guide the adaptation of the detector to the new distribution. Our design brings in two critical advantages. First, it can naturally handle any type of drift that degrades the performance of DL-based malicious traffic detectors. Second, it is model-agnostic and can be applied to both supervised and unsupervised DL-based malicious traffic detectors without any modifications to the underlying DL models.

Specifically, there are three modules to achieve the idea, as shown in Fig. 1. The traffic correlation measurement module accurately measures the correlations between high-dimensional and heterogeneous testing traffic samples via an ensemble clustering algorithm. Next, the label inference module formulates a multi-objective optimization problem to select a small set of highly representative testing samples for manual labeling. The optimization process jointly maximizes distributional coverage, i.e., how well the selected samples cover the whole testing distribution, and minimizes the labeling cost. The chosen samples are manually labeled by security analysts and leveraged to infer other testing samples' labels based on the well-measured traffic correlations. Further, the drift detection and adaptation module detects drift by checking the differences between the inferred labels and the predictions of the DL-based detector. This module also applies a novel incremental training loss to make the DL-based detector learn the new testing distribution while reducing the negative impacts of label noises (originating from label inference) and spatiotemporal imbalance. Finally, the DL-based detector can accurately classify future traffic. Note that our framework can also be applied to traditional ML-based malicious traffic detectors, we discuss it in Appendix E.

B. Measuring Traffic Correlation

The traffic correlation measurement module is responsible for measuring intrinsic correlations among testing traffic, to facilitate representative sample selection and accurate label inference. However, it is challenging to achieve this goal because: (i) The real network traffic is generated by diverse applications or attacks (e.g., web browsing, DDoS, or web attacks) that exhibit heterogeneous behaviors. (ii) The network traffic is often in high-dimensional feature vectors, e.g., the Kyoto [61] dataset has 542 features. These two characteristics make the traffic correlations vary in high-dimensional feature space, which are unpredictable and thus hard to capture. Existing traffic clustering methods only measure traffic correlations from limited feature dimensions, e.g., from flow interaction features to detect low-speed attack traffic [18] and from destination features to detect mobile traffic [63], failing to satisfy our requirements.

Traffic Correlation Mining. To accurately measure the correlations between the high-dimensional testing traffic data having heterogeneous behaviors, we apply the ensemble clustering method in this module. Ensemble clustering method can integrate multiple clustering algorithms that measure the correlations between network traffic from different aspects, preventing traffic with partial relevance (e.g., only in specific features) from being grouped into the same cluster. Specifically, we build the ensemble clustering method based on two classical clustering algorithms: K-means [23] and DBSCAN [55]. K-means and DBSCAN aggregate similar samples based on the Euclidean distance and the density of each sample, respectively. We set their hyper-parameters the same as the existing works [55], [39]. Given a testing dataset D_{test}, the method first utilizes K-means and DBSCAN algorithms to aggregate similar testing samples into clusters, respectively, and achieves the intersections of the clusters generated by these two algorithms as the results of ensemble clustering. That is, if a pair of samples are both in one cluster generated by K-means and another cluster generated by DBSCAN, they are in the same cluster of ensemble clustering.

Traffic Correlation Representation. We construct a normalized weighted adjacency matrix to record the traffic correlations captured by our ensemble clustering method while eliminating the negative impacts of varied cluster sizes on the correlations. Specifically, we construct a matrix denoted as $M \in \mathbb{R}^{|D_{\text{test}}| \times |D_{\text{test}}|}$, where each element M_{ij} represents

the correlation strength between testing sample X_i and X_j, where $i, j \in [0, |D_{\mathsf{test}}|)$. The matrix is computed by:

$$M_{ij} = \begin{cases} \frac{1}{dis(X_i, X_j)}, & \text{if } i \neq j \text{ and } c(X_i) = c(X_j) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $c(X)$ represents the cluster to which X belongs and $dis()$ computes the Euclidean distance between two instances. Then we normalize M symmetrically using its Laplacian form:

$$C = D^{-\frac{1}{2}} M D^{-\frac{1}{2}}, \quad (2)$$

where $D = diag(M\mathbf{1}_{|D_{\mathsf{test}}|})$ is a diagonal matrix, $\mathbf{1}_{|D_{\mathsf{test}}|}$ is an all-ones vector of length $|D_{\mathsf{test}}|$ and each diagonal element $D_{ii} = \sum_j M_{ij}$. The matrix C represents the traffic correlations between different testing samples and each element C_{ij} is equalient to $\frac{M_{ij}}{\sqrt{D_{ii} D_{jj}}}$. We can see that for the sample belonging to a large cluster, its corresponding diagonal element in D will be larger, thus reducing its correlations to other samples, i.e., has a smaller element in C. By contrast, the sample belonging to a smaller cluster will obtain a larger element in C because of its corresponding diagonal element in D becomes larger. Thus, this normalization design reduces the negative impacts of varied cluster sizes on the correlations, helping us to find the representative samples of all testing traffic more accurately.

C. Inferring Labels of Testing Traffic

The label inference module is in charge of selecting a small number of highly representative testing traffic samples for manual labeling and using them to infer the labels of the rest testing traffic based on traffic correlations. Intuitively, these samples should cover the entire distribution of the testing traffic, i.e., all regions within the feature space after clustering. Yet, in dynamic network environments, the traffic distribution may be extremely diverse [52], [30], leading to many dense and sparse regions with unpredictable shapes. Thus, it is difficult to locate a small number of highly representative network traffic samples that can effectively cover these regions.

Representative Sample Selection. To overcome the problem, we define a multi-objective optimization function that comprehensively balances the gains of labeling specific testing samples and the involved costs. In particular, the gains of selecting testing samples are distribution-aware and include two aspects: (i) To cover dense regions, selecting a sample that can be used to infer more other testing samples' labels obtains higher gains; (ii) Regarding sparse regions, choosing a sample that significantly deviates from the (feature-level) distribution of training traffic gets higher gains, as it is typically caused by rare and unknown network behaviors that deserve manual scrutiny, e.g., zero-day attacks or new network applications. Meanwhile, the cost is the increase of manual labeling overhead. Therefore, let $\mathbf{v}$ denotes an indicator vector of sample selection, where v_i indicates whether the i-th testing sample is selected (1 for selected and 0 for not) for manual labeling, the multi-objective optimization function is:

$$\min_{\mathbf{v}} : \mathcal{L}_{\mathsf{dense}} + \lambda_1 \mathcal{L}_{\mathsf{sparse}} + \lambda_2 \mathcal{L}_{\mathsf{lab}} + \lambda_3 \mathcal{L}_{\mathsf{sel}}, \text{ s.t. } \mathbf{v}_i \in [0, 1], \quad (3)$$

where:

$$\mathcal{L}_{\mathsf{dense}} = 1 - \frac{\sum_{i=1}^{|D_{\mathsf{test}}|} \min(z_i, 1)}{|D_{\mathsf{test}}|}, \quad (4)$$

$$\mathcal{L}_{\mathsf{sparse}} = (\mathbf{1}_{|D_{\mathsf{test}}|} - \mathbf{v}) \odot \mathsf{Conformal}(D_{\mathsf{train}}, D_{\mathsf{test}}), \quad (5)$$

$$\mathcal{L}_{\mathsf{lab}} = \frac{1}{|D_{\mathsf{test}}|} \parallel \mathbf{v} \parallel_1, \quad (6)$$

$$\mathcal{L}_{\mathsf{sel}} = \mathop{\mathbb{E}}_{v \in \mathbf{v}} [v \log v + (1 - v) \log(1 - v)]. \quad (7)$$

As shown in Eq.(3), our function consists of four major terms weighted by three configurable hyper-parameters λ_1, λ_2, λ_3 to accommodate different application demands. For example, we can set a higher λ_2 value when the cost of manual labeling is more expensive. We will discuss how to accelerate manual labeling in § V.

In detail, given a $\mathbf{v}$ vector, the first term $\mathcal{L}_{\mathsf{dense}}$ reflects how many other testing samples' labels would be inferred given the samples selected in $\mathbf{v}$. Based on the normalized traffic correlation matrix C, we follow the classical graph-based diffusion theory [75] and calculate $z_i \in \mathbf{z} = (I - \mu C)^{-1}\mathbf{v}$ which indicates how likely it is to derive the label of the i-th instance if the samples selected by $\mathbf{v}$ have labels. μ is a hyper-parameter ranging from 0 to 1 that controls the impact of correlations between testing samples in label inference. Then, we accumulate the possibility of label inference for all testing samples by calculating $\sum_{i=1}^{|D_{\mathsf{test}}|} \min(z_i, 1)$. It is better to get a $\mathbf{v}$ that can infer the labels of more other samples. The second term $\mathcal{L}_{\mathsf{sparse}}$ calculates the deviation in data distribution between the samples selected in $\mathbf{v}$ and the training dataset D_{train} via a popular conformal evaluation function [32], [8] denoted as $\mathsf{Conformal}$. The larger $\mathsf{Conformal}$ value means the deviation is more severe. We expect the samples selected in $\mathbf{v}$ having more severe deviations in data distribution such that more unknown network behaviors, e.g., zero-day attack, could be manually identified and labeled. Note that in the online deployment of our framework, the testing sets corresponding to previous time windows can be merged into the training dataset. The third term $\mathcal{L}_{\mathsf{lab}}$ measures the overhead of manual labeling, i.e., the number of selected samples, which should be reduced in the optimization. The last term $\mathcal{L}_{\mathsf{sel}}$ computes the entropy of the elements in $\mathbf{v}$ and is to make each element close to 0 or 1, i.e., determine to select one sample or not.

To solve this optimization function, we randomly initialize a $\mathbf{v}$ vector and then apply the gradient descent method [7]. When the algorithm converges or reaches a pre-defined number of iterations, the samples selected in $\mathbf{v}$ can represent both the majority of other testing samples with high correlations (i.e., in the dense regions within the feature space) and the outlier samples whose data distribution is significantly deviated (i.e., in the sparse regions), facilitating the inference of all testing samples' labels. The number of selected samples will be small, only incurring a slight manual labeling overhead. Finally, the

selected testing traffic samples are manually labeled, e.g., by security experts, to get their ground-truth labels.

Label Propagation. Given the fine-grained traffic correlation matrix and the manually labeled testing traffic samples, we infer the labels of other testing samples through a concise yet effective graph-based learning method, i.e., the label propagation algorithm [75], [28]. This method can efficiently infer the labels of other testing samples only using their correlation measurements with the labeled samples. By contrast, other graph-based label inference approaches such as belief propagation or graph neural network [13], [38] require more complicated information between testing samples, e.g., the temporal dependency or network topology, which are hard to extract in large-scale and dynamic networks and will undermine the efficiency of our framework in detecting drift.

Specifically, we create a matrix T of size $|D_{\mathsf{test}}| \times 2$ to record the testing samples' labels, where each row corresponds to one sample. The 0 and 1 column represents the probability of being normal and malicious, respectively. The element T_{ij} ($i \in [0, |D_{\mathsf{test}}|)$, $j \in \{0, 1\}$) in T is 1, if i-th testing sample is labeled as j, otherwise, is 0. Then, we calculate a pseudo-label matrix $\widetilde{T} = (I - \mu C)^{-1} T$ based on T and the normalized traffic correlation matrix C. The computation lets every sample iteratively spread its label (stored in T) to its neighbors (encoded in $\widetilde{T}$) until a global stable state is achieved [75], [28]. For the i-th testing sample X_i, if it is not manually labeled, its label $\hat{y}_i$ is inferred by $\hat{y}_i = \arg\max_j \widetilde{T}_{ij}$. We also assign this label a uncertainty factor $u_i = \max(\frac{\widetilde{T}_{i0}}{\sum_k \widetilde{T}_{ik}}, \frac{\widetilde{T}_{i1}}{\sum_k \widetilde{T}_{ik}})$, where $k \in \{0, 1\}$. If the selected samples have been manually labeled, their labels remain unchanged.

D. Learning Drifting Traffic

The drift detection and adaptation module identifies the conditional distribution shift (i.e., the indicator of concept drift) by comparing the inferred labels and the predicted labels (by the DL-based detector) of the testing traffic. When concept drift occurs, it incrementally trains the DL-based detector based on the current and historical labeled traffic to improve the detector's performance on subsequent testing traffic.

Drift Detection. We detect conditional distribution shift by computing $\Delta = \sum_{i=0}^{|D_{\mathsf{test}}|-1} u_i(\hat{y}_i \oplus f(X_i))$, where $f(X_i)$ is the DL-based detector's prediction result for X_i, $\hat{y}_i$ is the inferred label and u_i is the uncertainty factor. If Δ exceeds a threshold, we consider that concept drift occurs for the detector.

Drift Adaptation. Making the DL-based detector adapt to the drifting traffic based on their inferred labels needs to solve two challenges: First, manual labeling or label propagation may produce wrong labels in practice, i.e., label noises, making the detector learn incorrect data distribution. Second, in dynamic networks, the labeled testing traffic for adaptation may be of varied amounts at different times and have skewed class proportions, hindering the detector from accurately adapting to the new distribution. Existing drift adaptation techniques cannot handle these issues simultaneously, e.g., directly training on the labeled testing traffic [67], [43] and weight regularization [22], [14] are sensitive to label noises. To reduce the negative impacts of label noises and spatiotemporal imbalance. we propose a novel incremental training loss function that includes a self-calibration term and adaptive weights.

1) Self-Calibration against Label Noises. We design an entropy-based self-calibration term to reduce the negative impacts of label noises. The underlying rationale is that the output of a DL model typically has high entropy when the training data is with label noises [64], [51], which means that reducing the entropy of the DL-based detector's outputs can improve its robustness to label noises. Thus, we design a term H to measure the entropy of the DL-based detector's outputs

$$H(f(X)) = -p_0(X) \log p_0(X) + p_1(X) \log p_1(X), \quad (8)$$

where p_0 and p_1 are the probabilities of sample X being predicted as normal and malicious, respectively. For the supervised DL-based detector, p_0 and p_1 can be obtained from the outputs of the DL model's last layer via the softmax function. We will discuss how to extract these probabilities from unsupervised DL-based detectors later.

2) Adaptive Weights against Spatiotemporal Imbalance. To mitigate the negative impacts of spatiotemporal imbalance in incremental training, we introduce two adaptive weights: 1) ω_t is to make the DL-based detector learn more from recently labeled testing samples, while still learning some historical knowledge for preventing catastrophic forgetting. 2) ω_s is to assign different weights for samples of different categories to reduce the negative impact of skewed class proportions.

Assume on the i-th time window, we combine all labeled testing sets corresponding to the i-th and earlier time windows, i.e., $D^1_{\mathsf{test}}, ..., D^i_{\mathsf{test}}$, with the training set D_{train}, as the incremental training set D. τ_t is the latest time window and 0 is the time window of the original training set D_{train}. We set ω_t for the data collected on the i-th time window ($i \in [0, \omega_t]$) as $e^{-A(\tau_t - i)}$. ω_t is smaller on earlier time windows and A is a tunable parameter that controls the weight reduction speed. Besides, we set ω_s for the data on the i-th time window with j label as $\frac{1}{|N_{ij}|}$, where j is normal or malicious and $|N_{ij}|$ is the number of samples with j label on the i-th time window.

3) Incremental Training Loss Function. Finally, the incremental training loss function is defined as:

$$\mathcal{L}(D) = \sum_{(X,y) \in D} (\omega_t \cdot \omega_s \cdot \ell(X, y) + \lambda_4 H(f(X))), \quad (9)$$

where ℓ represents the original loss function of the DL-based detector and λ_4 is the hyper-parameter for controlling the importance of the entropy. We can utilize this function and D to make any supervised DL-based detector adapt well to the shifted data distribution while reducing the negative impacts of label noises and spatiotemporal imbalance.

Application to Unsupervised DL-based Detectors. Since unsupervised DL-based detectors only use normal traffic for training, we define a modified loss function to make them effectively learn both normal and malicious samples in the incremental training process. Its core idea is to increase the predicted probabilities of the unsupervised detector for normal samples while decreasing the probabilities for malicious ones.

Specifically, we denote the original loss function of the unsupervised DL-based detector as $\ell_\varphi(X)$, which can only output the probability of predicting X as normal. Then, the modified loss function can be defined as:

$$\ell' = \mathsf{ReLU}(B - y\ell_\varphi(X) + (1-y)\ell_\varphi(X)), \qquad (10)$$

where y is the label of X (0 for normal and 1 for malicious), ReLU is the rectified linear activation function [21] that limits the lower boundary of the loss, and $B > 0$ is a pre-determined constant and can be set as twice the classification threshold of the detector [14]. Finally, we combine this function with the self-calibration term and adaptive weights as the incremental training loss function for unsupervised detectors: $\mathcal{L}'(D) = \sum_{(X,y)\in D}(\omega_t \cdot \omega_s \cdot \ell'(X,y) + \lambda_4 H(f'(X))$, where p_0 and p_1 in $H(f'(X))$ can be achieved by: $p_0 = \frac{B-\min(\ell_\varphi(\boldsymbol{x}),B)}{B}$ and $p_1 = \frac{\min(\ell_\varphi(\boldsymbol{x}),B)}{B}$.

IV. Evaluation

A. Experimental Setup

Public Datasets. We use the following public datasets. They are collected in large real-world or simulated network environments and have been widely evaluated by existing anti-concept drift studies [8], [70], [12], [22], [12], [67].

- *Kyoto 2006+ (Kyoto)* [61] stores the network traffic collected by multiple honeypots inside and outside Kyoto University, from 2006 and 2015. It extracts flow-level statistical features from the traffic, e.g., the max packet size in a flow.
- *CIC-IDS2017 (CIC)* [56] stores the network traffic collected from a large network over 5 days. It extracts flow-level statistical features from the traffic. Since there is little malicious traffic on Monday and Thursday of the dataset, we only use the other three days' data for evaluation.
- *RWDIDS2022 (RW)* [67] contains the network traffic collected from a network having nine IoT devices for six days, i.e., from 12-09 to 12-17. It extracts packet-level features, e.g., the packet length, from the traffic. We only use the data collected from 12-12 to 12-16 for evaluation because there is no malicious traffic on other days.

Synthetic Datasets. We construct four synthetic datasets to evaluate the performance of our framework under stealthy and non-stealthy, normal and malicious drifting samples individually. Their statistics are provided in Appendix A.

- *Dataset of Stealthy Normal Drifting (S-Normal)*. We use the traffic of the first three years in Kyoto as the background. For each normal sample in the later seven years, we check whether it is a stealthy normal drifting traffic sample using the method discussed in § II-A. Then, we insert the drifting samples around the middle of the time span of the background to create this synthetic dataset. Note that we do not insert drifting samples at the later time periods. This allows the data distributions corresponding to the initial and later time periods unchanged, such that we can evaluate our framework's resilience to catastrophic forgetting [34].
- *Dataset of Stealthy Malicious Drifting (S-Malicious)*. This dataset is created in a similar way as the S-Normal. We create it based on the CIC dataset because we find that the malicious drifting samples in this dataset are more stealthy than those in other datasets. We use the traffic on the third day of CIC as the background and extract stealthy malicious drifting samples from the fifth day's traffic.
- *Dataset of Non-Stealthy Normal Drifting (NS-Normal)*. We use the randomly shuffled normal and malicious traffic of the RW dataset as the background. This mitigates distribution shifts in the background traffic. Then, we select the normal samples in the MAWI_2020.06.10 [5] and the Mirai [44] datasets as the non-stealthy normal drifting samples, since the traffic's distribution of different public datasets is usually inconsistent. We insert them into the background in the same way as the S-Normal. The iP2V tool [67] is used to extract packet-level features from the traffic.
- *Dataset of Non-Stealthy Malicious Drifting (NS-Malicious)*. It is created in a similar way as the NS-Normal. We randomly shuffle the normal and malicious traffic of the RW dataset as the background and select the malicious samples from the ML Adversarial Attack [67] and the Mirai [44] datasets as the non-stealthy malicious drifting samples.

Data Splitting and Evaluation Strategy. To demonstrate that `Chameleon` can continuously combat concept drift, we split each public dataset into a training window and a sequence of testing windows. Specifically, we randomly sample 50,000 samples from the first year traffic of Kyoto, the first half day traffic of CIC, and the first day traffic of RW, to serve as the training window, respectively. Then, we split the subsequent traffic of Kyoto, CIC, and RW into half-year, quarter-day, and half-day time intervals, respectively, and randomly sample 50,000 traffic samples from each time interval of each dataset as a testing window of this dataset. Besides, since the synthetic datasets contain traffic samples from multiple sources, we use a fixed window size of 50,000 samples to split each synthetic dataset, where the first window is for training and others are for testing. To avoid sampling bias, we repeat this data split process three times and report the average experimental results.

We adopt the non-stop evaluation strategy [12] to conduct experiments. In particular, for one dataset, we train the DL-based detector using the initial training set and take the following steps: 1) use the detector to test the data in the current testing window, 2) apply our framework to detect and adapt to concept drift in the current testing window, and 3) move to the next window and repeat these steps until the end of windows. We denote the initial training window as T0. To better show performance variation in a limited space, we accumulate the results of consecutive windows at the granularity of year, half day, and day, for Kyoto, CIC, and RW, respectively. We denote the result of accumulated window as T{N}, where N is the accumulated window's chronological order (starting from 1).

Baselines. We select multiple state-of-the-art (SOTA) anti-concept drift approaches as baselines.

- **Baselines for supervised DL-based detector.** We choose four SOTA methods designed for supervised DL-based malware (traffic) detectors: TRANS [8], CADE [70], HICL [12],

Table II: The performance of MLP and KitNet detectors when using Chameleon and baselines on public datasets.

DL-based Detector	Method	Kyoto			CIC			RW		
		F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)
MLP (Supervised)	OFFLINE	61.36	67.31	60.92	21.78	62.73	16.32	85.72	93.60	87.27
	TRANS [8]	72.05	75.93	72.45	43.53	69.05	33.33	92.40	96.56	92.98
	CADE [70]	71.12	76.34	71.28	51.67	67.62	42.86	92.59	95.70	92.59
	HICL [12]	70.43	81.80	71.82	63.68	77.10	57.08	87.56	90.96	88.87
	INSOMNIA [6]	58.92	74.74	58.10	55.59	73.18	48.97	77.83	79.47	80.37
	Chameleon	**80.88**	**88.03**	**80.58**	**70.71**	**96.35**	**65.20**	**92.86**	**96.56**	**93.38**
KitNet (Unsupervised)	OFFLINE	25.48	38.32	23.40	52.38	74.17	48.63	49.98	46.79	48.67
	OWAD [22]	48.28	62.80	47.59	54.40	79.97	50.85	64.93	72.58	65.14
	Chameleon	**76.05**	**86.60**	**76.43**	**75.56**	**92.62**	**72.34**	**84.68**	**91.77**	**85.98**

and INSOMNIA [6]. Since TANS and CADE do not have a drift adaptation design, we retrain the detector based on the drifting samples and the original training data, which is the same as the existing study [22].

- **Baseline for unsupervised DL-based detector.** We choose the SOTA anti-concept drift approach designed for unsupervised DL-based anomaly detectors, i.e., OWAD [22].

We describe the implementation of these baselines in Appendix B. Besides, we apply the OFFLINE method as another baseline, i.e., not perform any drift detection and adaptation.

DL-based Malicious Traffic Detectors. We select two representative DL-based malicious traffic detectors: a Multi-Layer Perceptron (MLP) [12] and KitNet [44]. MLP is a representative supervised DL model and can detect various kinds of network attacks [70], [72]. KitNet [44] is a SOTA unsupervised DL-based detector. Further, to demonstrate the applicability of our framework to diverse DL-based detectors, we introduce several detectors based on sequence-based and graph-based DL models, including the supervised RNN [68] and GIN [57], and the unsupervised LSTM [15] and GNN [66]. We show the implementation information of these detectors in Appendix C.

Implementations. We implement Chameleon with over 5,000 lines of code using Python 3.8.8, PyTorch [47] and Scikit-Learn [48]. We perform a grid search to set the hyper-parameter λ_1, λ_2, λ_3, and λ_4 and they are set to 1, 10, 1, and 0.05, respectively. The configurations of other hyper-parameters are provided in Appendix D.

Metrics. We choose the widely used F1-score (F1) and Accuracy to evaluate the performance. We also introduce the Area Under Time (AUT) [49] to assess the overall performance over a long time. This metric calculates the area under the performance (F1) curve across various time windows.

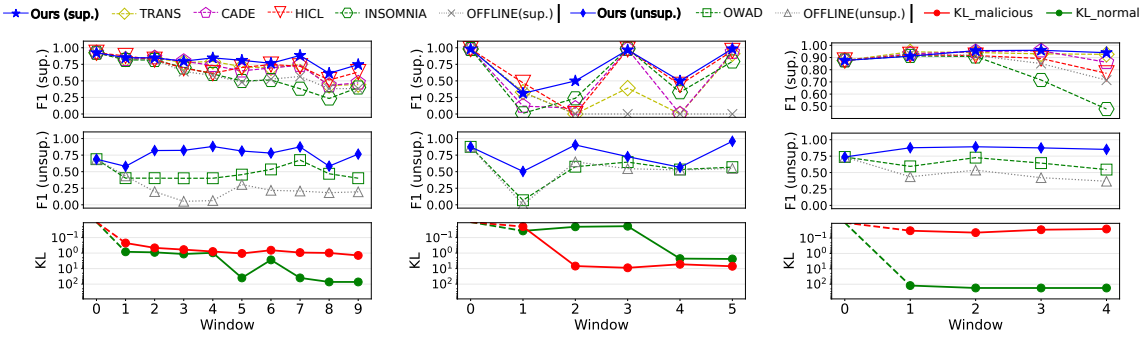
B. Overall Detection Performance

We enhance the DL-based detectors with our framework and the baselines individually and evaluate their performance under concept drift. For fair comparison, we limit the labeling overhead on each testing window to 1%, i.e., only allowing using 1% samples' true labels at most in each testing window. To avoid the experimental bias incurred by different concept drift detection thresholds, all baselines and Chameleon are set to perform drift adaptation on each testing window.

Performance on Public Datasets. Table II shows the overall performance of the supervised MLP and the unsupervised KitNet detectors on the public datasets after we enhance them with our framework and all baselines individually. The F1 score and accuracy are averaged across all testing windows. We can see that the supervised MLP enhanced with Chameleon obtains the best AUT scores on the three datasets, outperforming the MLP enhanced with baselines (except OFFLINE) by 18.00% on average. Also, the unsupervised KitNet enhanced with Chameleon achieves the best AUT scores, surpassing the KitNet augmented with OWAD by 43.51% on average. These improvements indicate that Chameleon is more effective in maintaining the detection performance of both supervised and unsupervised DL-based detectors under concept drift. The possible reason for this enhanced capability is Chameleon's unique ability to handle both stealthy and non-stealthy drift which might concurrently exist in diverse network environments yet are not well tackled by existing arts.

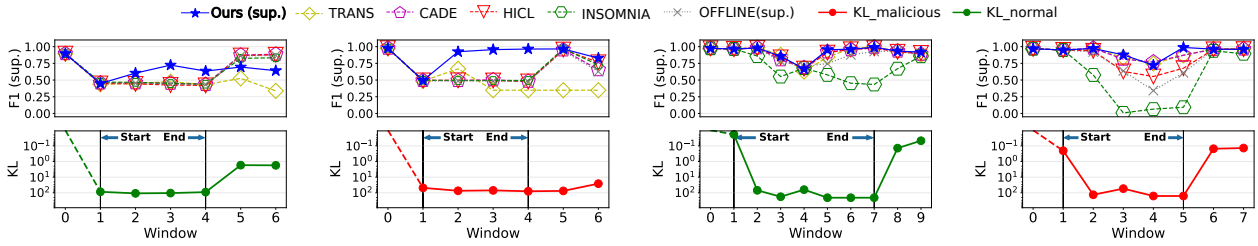
Figure 2 shows the F1 of the enhanced MLP and KitNet on different windows of the public datasets. We use a well-known information-based measure, i.e., Kullback-Leibler (KL) divergence, to measure the difference between the (normal and malicious) data distributions of each testing window and the training window. This metric reflects the severity level of concept drift. A higher KL divergence means a higher level of concept drift. We can find that our framework can adapt to drift more effectively than baselines in both supervised and unsupervised settings. For instance, on the T1 of the Kyoto dataset, all F1 scores of the DL-based detectors descend. Yet, when using Chameleon, the detectors' performance achieves the best on T2. Besides, Chameleon maintains a more stable performance across different windows than the baselines. For instance, on the T2-T7 windows of the Kyoto dataset, the F1 of the KitNet enhanced with Chameleon remains in the range of 78.07% and 88.06%, while the F1 of the KitNet enhanced with OWAD fluctuates between 40.04% and 67.47%. Overall, these results indicate that our framework can maintain the performance of DL-based detectors under varied network environments with different severity levels of concept drift.

Case Study. We take an in-depth study of a kind of stealthy malicious drifting traffic detected in the CIC dataset by our framework. We analyze the plaintext payload of these malicious traffic and find they (891 flows in total) are denial of service attack traffic based on the HTTP protocol, which requests the resources in the root directory of the target server and sends



(a) F1 on the Kyoto dataset (b) F1 on the CIC dataset (c) F1 on the RW dataset

Figure 2: F1 of MLP and KitNet detectors when using Chameleon and baselines on each testing window of the public datasets.

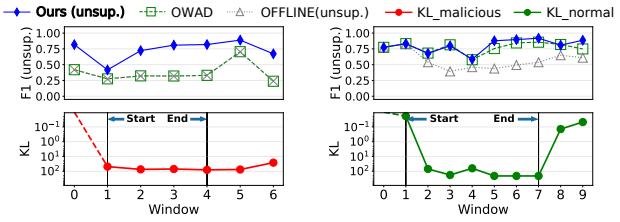


(a) F1 on the S-Normal dataset (b) F1 on the S-Malicious dataset (c) F1 on the NS-Normal dataset (d) F1 on the NS-Malicious dataset

Figure 3: F1 of the MLP detector when using Chameleon and the baselines on each testing window of the synthetic datasets.

a large number of ACK packets to exhaust the target server's resources. They are happened from 2017-07-07 15:58:00 to 2017-07-07 16:14:00. By measuring the feature-level distance via the method aforementioned in § II-A, we identify over 500 instances of normal training traffic having the most similar features to the malicious traffic, whose duration are from 2017-07-04 10:17:00 to 2017-07-04 10:54:00. These training traffic are HTTP requests sent by a Windows Vista server to request specific JPEG images from opencdn.jomodns.com. They also contain a lot of retransmission and ACK packets due to network jitters. It can be seen that the behavior pattern of these training traffic is very similar to that of the above malicious traffic, resulting in more than 50 similar features, such as "Bwd Packet Length Max", "Bwd IAT Total", "Fwd PSH Flags". Therefore, the baselines (e.g., HICL) that solely check the marginal distribution of features fail to detect these stealthy malicious drifting traffic, while our framework identifies them by inferring the conditional distribution of testing data.

Performance on Synthetic Datasets. With the synthetic datasets, we evaluate the detection performance of the enhanced MLP and KitNet under different types of concept drift and show their results in Tab. III and IV, respectively. Note that only stealthy malicious drifting samples (i.e., the malicious testing samples have similar features to the normal training samples) and non-stealthy normal drifting samples (i.e., the normal testing samples whose features are not similar to the normal training samples) may impair the detection performance of the unsupervised detector. Thus, we only use



(a) F1 on the S-Malicious dataset (b) F1 on the NS-Normal dataset

Figure 4: F1 of KitNet when using Chameleon and the baselines on each testing window of the synthetic datasets.

the S-Malicious and NS-Normal dataset to evaluate KitNet.

As shown in Tab. III and IV, Chameleon performs best in all cases, while the baselines are less effective in handling stealthy drift. Specifically, Chameleon improves the AUT of the (OFFLINE) MLP by 0.2389 on average on the S-Normal and S-Malicious datasets, while the baselines improve the AUT of the (OFFLINE) MLP by 0.0186 on average. The improvement incurred by Chameleon is about 12.84 times the average improvement caused by the baselines. On the NS-Normal and NS-Malicious datasets, the MLP's AUT improvement incurred by Chameleon is also higher than the average improvement caused by the baselines, i.e., about 1.437 times. For the KitNet, OWAD makes little improvement on the S-Malicious dataset while Chameleon achieves 0.1853 improvement on AUT. Chameleon is 1.148 times better than

Table III: The performance of MLP detector when using Chameleon and the baselines on the synthetic datasets.

Method	S-Normal			S-Malicious			NS-Normal			NS-Malicious		
	F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)
OFFLINE	62.87	60.09	66.51	63.90	81.81	75.45	87.87	91.98	87.21	79.29	91.91	76.81
TRANS[8]	52.72	53.16	59.75	50.33	65.64	62.27	90.50	93.33	90.30	91.09	95.76	90.23
CADE[70]	63.65	60.85	67.29	65.04	82.23	76.09	91.00	94.09	90.70	90.93	94.70	90.11
HICL[12]	63.07	60.05	66.60	66.57	82.97	77.01	91.02	94.04	90.70	82.96	92.88	81.01
INSOMNIA[6]	63.06	59.80	67.52	66.38	83.18	76.96	70.22	72.81	67.83	56.14	87.30	50.86
Chameleon	**66.76**	**74.44**	**74.11**	**87.26**	**92.65**	**90.83**	**92.00**	**94.69**	**91.70**	**92.52**	**96.35**	**91.96**

Table IV: The performance of the KitNet detector when using Chameleon and baselines on the synthetic datasets.

Method	S-Malicious			NS-Normal		
	F1(%)	Acc(%)	AUT(%)	F1(%)	Acc(%)	AUT(%)
OFFLINE	37.20	71.57	61.49	57.20	59.16	55.90
OWAD[22]	37.32	71.61	61.55	77.03	**89.20**	77.14
Chameleon	**73.48**	**85.22**	**80.02**	**80.55**	86.00	**80.29**

AUT when using OWAD on the NS-Normal dataset.

The performance of the enhanced MLP and KitNet detectors on different windows of the synthetic datasets is shown in Fig. 3 and 4, respectively. On these figures, we mark the windows which we start and end inserting drifting samples and show the distribution difference (i.e., via the KL divergence) between each testing window and the training window. We can see that when the drifting samples start to appear (e.g., T1 of the S-Malicious dataset), the detectors' performance degrade. However, after our framework makes the detectors adapt to these drifting samples, the detectors' performance significantly improves (e.g., T2-T4 of the S-Malicious dataset). Overall, the detectors enhanced with our framework achieve the best performance on most testing windows of different datasets, showing that our framework can handle both the stealthy and non-stealthy drift with varying levels of severity.

The results in Fig.3 and 4 also demonstrate the resilience of our framework to catastrophic forgetting [34], i.e., DL models forget useful knowledge about the historical distribution after adapting to the new distribution. Recall that we do not insert drifting samples at the later periods of each synthetic dataset. Thus, on most of the first windows after stopping drifting sample insertion (i.e., T5 of the S-Normal, T8 of the NS-Normal, and T6 of the NS-Malicious), the data distribution becomes similar to the background traffic. On these windows, the performance of the detectors with our framework becomes close to their performance on T0 (i.e., the training set). These results indicate that even after multiple drift adaptations, our framework retains enough knowledge about the historical distribution for the DL-based detectors.

Applicability to Different DL-based detectors. We evaluate the applicability of our framework to DL-based detectors with diverse architectures, including two detectors using sequence-based DL models (supervised RNN [68] and unsupervised LSTM [15]), and two detectors using graph-based DL models (supervised GIN [57] and unsupervised GNN [66]). We enhance them with our framework and the baselines individually and show their AUT scores on the Kyoto dataset in Tab. V.

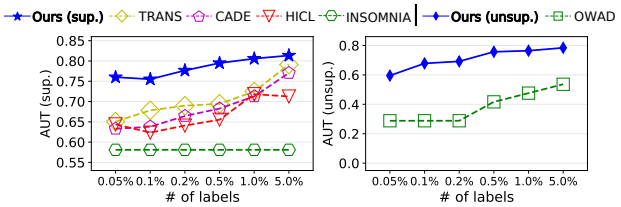


Figure 5: AUT of the MLP and KitNet detectors with Chameleon and baselines on the Kyoto dataset under different labeling overhead budgets.

Note that FC (i.e., Fully Connected) refers to MLP and KitNet in the supervised and unsupervised setting, respectively. Overall, our framework significantly improves the performance of all detectors compared with the original detectors (i.e., OFFLINE) and our improvements are much higher than those of the baselines. For the supervised and unsupervised sequence-based DL detectors, the AUT improvement of our framework is 2.820 and 2.939 times the improvement induced by the baselines on average, respectively. For the supervised and unsupervised graph-based DL detectors, the improvement of our framework is 3.674 and 2.680 times the improvement induced by the baselines on average, respectively. These results demonstrate our framework is applicable to diverse DL-based malicious traffic detectors.

C. Overhead Evaluation

We evaluate the impact of different labeling overhead budgets, i.e., the maximum proportion of samples whose ground truth labels can be used in each testing window. We use the Kyoto dataset for evaluation and the other settings are the same as in § IV-B. Since the optimization function in our framework, i.e., Eq.(3), may identify more samples that require ground truth labels than the budget, we randomly select some samples with the same number as the budget from the samples identified by Eq.(3), which is the same as OWAD [22]. Then, our framework uses the ground truth labels of these selected samples to infer the labels of other testing data.

We enhance MLP and KitNet with Chameleon and the baselines under different labeling overhead budgets. As shown in Fig. 5, the performance of our framework decreases slightly as the labeling overhead budget declines. For instance, when the labeling overhead budget is 0.05%, the MLP enhanced with Chameleon obtains an AUT of 76.01%, only a slight decline from the 80.58% AUT of the 1.0% budget. Note that the 0.05%

Table V: AUT(%) of different DL-based detectors when using Chameleon and the baselines on the Kyoto dataset.

DL-based Detector	Method	Kyoto FC	Kyoto Sequence	Kyoto Graph	CIC FC	CIC Sequence	CIC Graph	RW FC	RW Sequence	RW Graph
Supervised	OFFLINE	60.92	61.20	64.08	16.32	17.44	/†	87.27	76.18	83.40
	TRANS[8]	72.45	66.01	68.58	33.33	18.23	/	92.98	93.83	92.94
	CADE[70]	71.28	63.77	67.63	42.86	61.11	/	92.59	93.86	93.80
	HICL[12]	71.82	67.76	65.23	57.08	42.33	/	88.87	93.35	92.17
	INSOMNIA[6]	58.10	41.27	58.78	48.97	50.21	/	80.37	61.74	71.67
	Chameleon	**80.58**	**71.42**	**72.30**	**65.20**	**83.93**	/	**93.38**	**94.06**	**94.36**
Unsupervised	OFFLINE	23.40	20.20	21.22	48.63	54.31	/	48.67	40.38	46.21
	OWAD[22]	47.59	32.74	37.06	50.85	61.92	/	65.14	56.14	61.98
	Chameleon	**76.43**	**67.30**	**66.90**	**72.34**	**71.97**	/	**85.98**	**81.15**	**85.24**

†CIC does not provide the information for graph construction, e.g., the IP and port, thus it cannot be used to evaluate graph-based detectors.

Table VI: F1 and AUT of the MLP detector with Chameleon variants on the Kyoto dataset. RMeas, LInf, and DAda are the Traffic Correlation Measurement Module, the Label Inference Module, and the Drift Detection and Adaptation Module.

Variant of Chameleon	F1(%) T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	Avg. F1(%)	AUT(%)
K-Means + LInf + DAda	92.50	85.86	80.43	77.68	83.20	79.08	77.57	86.80	56.65	70.05	78.98	78.73
DBSCAN + LInf + DAda	92.50	82.76	65.65	73.99	59.67	53.06	56.08	53.76	47.76	41.79	62.70	62.21
GMM with DBSCAN + LInf + DAda	92.50	83.13	82.67	69.50	77.05	68.23	67.30	84.22	53.34	69.79	74.77	74.07
K-Means with GMM + LInf + DAda	92.50	85.44	85.76	75.40	81.32	79.95	71.82	85.02	55.23	71.51	78.40	77.99
Fare [39] + LInf + DAda	92.50	84.20	82.00	80.95	85.61	79.04	78.23	83.77	54.83	72.78	79.39	79.03
RMeas + LInf without $\mathcal{L}_{\text{den.}}$ + DAda	92.50	84.70	71.95	74.73	79.16	77.95	73.35	83.18	53.84	71.37	76.27	75.64
RMeas + LInf without $\mathcal{L}_{\text{spar.}}$ + DAda	92.50	86.98	84.05	73.80	79.20	78.74	72.95	85.90	55.11	71.88	78.11	77.66
RMeas + LInf without $\mathcal{L}_{\text{den.}}$ and $\mathcal{L}_{\text{spar.}}$ + DAda	92.50	83.96	79.86	76.45	82.19	78.75	72.34	82.39	57.10	72.47	77.80	77.28
RMeas + LInf without label prop. + DAda	92.50	84.01	84.45	70.21	70.73	71.92	70.06	71.08	51.66	66.25	73.29	72.61
RMeas + LInf + DAda without ω	92.50	83.37	77.70	70.80	78.00	72.71	78.38	88.89	58.81	71.89	77.31	76.76
RMeas + LInf + DAda without $H(\cdot)$	92.50	83.30	80.49	76.54	81.65	68.71	75.56	85.21	54.94	71.47	77.04	76.49
RMeas + LInf + DAda without ω and $H(\cdot)$	92.50	83.37	77.70	70.80	78.00	72.71	75.53	89.50	56.75	70.34	76.72	76.20
RMeas + LInf + weight reg. [22]	92.50	83.42	63.48	71.50	80.95	80.33	82.51	86.38	55.43	70.53	76.70	76.17
Original Chameleon (RMeas + LInf + DAda)	92.50	85.02	84.78	79.77	84.70	81.03	76.54	88.40	61.45	74.57	**80.88**	**80.58**

labeling overhead budget is equivalent to manually labeling only 50 samples per year on the Kyoto dataset, which is easy to realize in practice. More crucially, the baselines require more labeling overhead budget to approach the performance of our framework. For MLP, the baselines necessitate a high budget of about 5% to approach the same performance as our framework at a budget of 0.05% (i.e., 76.01%), showing a 100x reduction in labeling overhead of our framework.

D. Evaluating Individual Components

Evaluation of the Ensemble Clustering Method. We create five variants of our framework by replacing our ensemble clustering method with two single clustering algorithms: K-Means [23] and DBSCAN [55], and three ensemble clustering algorithms: GMM [26] with DBSCAN, K-Means with GMM, and Fare [39]. We evaluate these variants individually based on the MLP detector and the Kyoto dataset and show their performance in Tab. VI. Overall, the MLP with the original Chameleon achieves the best AUT score, outperforming the MLP with other variants by 8.30% on average, showing that our ensemble clustering method is more suitable for accurately measuring the correlation between heterogeneous and high-dimensional network traffic.

Evaluation of the Label Inference Design. We create four other variants of our framework by removing the key components in our label inference design individually, including the $\mathcal{L}_{\text{dense}}$ term, the $\mathcal{L}_{\text{sparse}}$ term, both of the terms, or the label propagation method. We do not remove the $\mathcal{L}_{\text{lab}}$ and $\mathcal{L}_{\text{sel}}$ terms because they are indispensable for representative sample selection. As shown in Tab. VI, the performance of the MLP with the original Chameleon exceeds the MLP with these variants and the average AUT improvement is about 6.31%, indicating that the $\mathcal{L}_{\text{dense}}$ and $\mathcal{L}_{\text{sparse}}$ terms and the label propagation method contribute to accurately inferring the labels of all testing traffic. Besides, the MLP with the variant without the label propagation method achieves the lowest average F1 (about 73.29%) and AUT (about 72.61%), about 9.38% and 9.89% lower than those of the MLP with the original Chameleon, suggesting that acquiring the label of a large amount of testing data is essential for drift adaptation.

Evaluation of the Incremental Training Loss. We create four additional variants of our framework by removing the components in our incremental training loss, i.e., the adaptive weights ω and the entropy-based self-calibration design $H(\cdot)$, or replacing the loss with other drift adpation method, i.e., the weight regularization method [22]. As shown in Tab. VI, the average F1 and AUT of the MLP with the original Chameleon is the best, showing average improvements of about 5.12% and 5.46% over the MLP with these variants, respectively. Besides, the performance of the weight regularization variant is inferior to other variants because it ignores the label noise and spatiotemporal imbalance issues in the testing traffic. These results indicate that our entropy-based self-calibration design and the adaptive weights can mitigate

Table VII: AUT of the MLP detector when using Chameleon on the Kyoto dataset under different hyper-parameter settings.

Hyper-Parameter	Value Range	AUT(%)
λ_1	$[10^{-1} \sim 10]$	[75.41 $\sim$ 80.58]
λ_2	$[10^{-1} \sim 10]$	[76.89 $\sim$ 80.58]
λ_3	$[10^{-1} \sim 10]$	[75.72 $\sim$ 80.58]
λ_4	$[10^{-3} \sim 1\ ]$	[76.93 $\sim$ 80.58]

Table VIII: F1 and AUT of the MLP detector when using Chameleon on Kyoto under varied drift detection thresholds.

Drift Detection Threshold	Avg. F1(%)	AUT(%)
10%	74.62	73.98
5.0%	76.89	76.41
1.0%	77.04	76.49
Drift Adaptation Per Testing Window	80.88	80.58

the negative impacts of the label noises and spatiotemporal imbalance in the testing traffic.

E. Hyper-parameter Sensitivity

We evaluate the impacts of different hyper-parameter values, including λ_1 (weight of data distribution deviation), λ_2 (weight of human labeling overhead), λ_3 (weight of sample selection determinism) in Eq.(3), and λ_4 (weight of prediction entropy) in Eq.(9), on the performance of our framework. In particular, we set a value range of $[10^{-1}, 10]$ and $[10^{-3}, 1]$ for λ_1-λ_3 and λ_4, respectively. For each hyper-parameter, five different values are uniformly sampled in a logarithmic scale from its value range. When evaluating one hyper-parameter, other hyper-parameters are set to their original values in § IV-A. All experiments are conducted based on the MLP and the Kyoto dataset and other settings are the same in § IV-B. As shown in Tab. VII, the performance of the MLP enhanced by our framework under different hyper-parameter settings varies slightly, e.g., the change of the AUT is in the range of 0.0365 to 0.0517. This indicates that the performance of our framework is not sensitive to hyper-parameter choices.

We also evaluate the impacts of different drift detection thresholds. As shown in Tab. VIII, the performance of Chameleon slightly degrades as the threshold increases. This is because a higher threshold requires more drifting samples in the testing window to trigger drift adaptation. A higher threshold means less manual labeling overhead. Thus, in practice, we can make a trade-off between the manual label overhead and the detection performance. Moreover, even using a higher detection threshold, our framework still outperforms the baselines that perform drift adaptation per window, e.g., the MLP with our framework achieves the F1 of 74.62%, while those of the baselines that adapt to drift per window are in the range of 58.92% to 72.05% (Tab. II). These results indicate that our framework can learn the drifting distribution more accurately and thus maintain better performance over time.

V. Discussions

Traffic Labeling Acceleration. As validated in § IV-C, our framework only requires a very small percentage of testing traffic in each time window to be manually labeled, e.g., about 0.5% traffic data per year to achieve an AUT of 79.49% on the Kyoto dataset. We argue that selecting 0.5% traffic samples per window for manual labeling is adequate for our framework in most cases because the samples selected by our framework are highly representative to cover all data distribution regions. In extreme cases, e.g., a high-speed network with highly dynamic behaviors, we may need to set a smaller time window for our framework to handle concept drift in time, increasing the total manual labeling overhead. To accelerate traffic labeling, we can utilize online threat analysis tools [3] that identify known attack traffic quickly or apply the crowdsourcing strategy [2], i.e., invite many people for traffic analysis and ensemble their prediction results. We leave the application of these techniques in our framework as future work.

Computation and Storage Overhead. The computational overhead of our framework is lower than existing approaches. For instance, on the Kyoto dataset with the MLP-based detector, our framework requires about 165.8s to detect and adapt to drift (excluding the labeling time), while TRANS, CADE, and HICL require 398.5s, 708.5s, 1027.4s, respectively. This advantage is mainly because our framework does not need to train an auxiliary model (i.e., CADE and HICL) or iterate over training samples multiple times to search optimal detection threshold (i.e., TRANS). Besides, our framework requires a relatively larger storage space than existing arts because it stores the inferred labels of all testing traffic. For instance, on the Kyoto dataset, our framework needs about 1784MB space, while existing approaches require about 200MB. This storage overhead is acceptable in practice, e.g., storing 100GB data on the cloud only needs 2 dollars per month [1].

Applicability to Other Security Domains. While Chameleon is designed and evaluated in the context of malicious traffic detection, its underlying principles are broadly applicable across various security domains that involve data distribution shifts and limited labeling budgets. For example, the features of malware samples (e.g., API call sequences, bytecode) may change over time due to new obfuscation techniques, while their malicious nature remains [70], [74], [36]. Chameleon could help identify and adapt to these new malware variants. We will explore the applicability of our work to other security domains by designing domain-specific data correlation measurement methods in future work.

Integration with Other Model Adaptation Methods. While Chameleon includes an adaptation approach via incremental fine-tuning, its modular design allows for integration with more advanced model adaptation techniques. One promising direction is to identify drift-invariant features between drifting traffic and historical data, enabling the model to learn more robust representations [36].

VI. Related Work

DL-based Malicious Traffic Detection. Existing DL-based malicious traffic detectors [44], [15], [68], [18], [31], [20], [62], [52] are built upon diverse DL models. Specifically, KitNet [44] utilizes an ensemble of autoencoder to perform unsupervised malicious traffic detection. Zerowall [62] and Deeplog [15] use recurrent neural networks to learn the temporal features and detect long-duration attacks. Some recent arts apply novel DL models to improve their robustness under low-quality training data [52] or low-speed attacks [18], yet they cannot resist the stealthy drifting traffic.

Concept Drift Detection. Existing concept drift detection methods [32], [8], [70], [12], [22], [25], [54] mainly focus on identifying feature-level distribution shifts from training to testing traffic. They utilize different methods to measure the distribution shift, e.g., the softmax confidence scores [25], the likelihood of generative models [54], [70], [12], the conformal evaluation [32], [8], and the Kullback-Leibler (KL) divergence. However, they cannot detect stealthy drifting traffic.

Concept Drift Adaptation. In general, there are three kinds of widely used concept drift adaptation methods. The first is the re-training strategy [6], [12], [49], i.e., re-train the DL model using the original training and the labeled new data. The second is the adaptive ensemble learning method [67], [43], [33], i.e., train an individual classifier based on the data in each time window and ensemble all classifiers' predictions. Lastly, the weight regularization method [22], [14] trains DL models using new data while restricting the change of important parameters to prevent catastrophic forgetting. However, none of them can make both supervised and unsupervised DL-based detectors accurately adapt to the new data distribution under label noises and spatiotemporal imbalance.

VII. Conclusion

In this paper, we devise `Chameleon`, a novel active learning framework that efficiently detects and continuously learns stealthy drifting traffic with a small amount of manual labeling effort. `Chameleon` involves an ensemble clustering based traffic correlation module that mines fine-grained correlations between high-dimensional and heterogeneous traffic, a label inference module that selects representative samples in a distribution-aware manner and infers the testing traffic's labels, and a drift detection and adaptation module that makes DL-based detectors learn new distribution under label noises and spatiotemporal imbalance. Experimental results indicate that, under stealthy drifting traffic, `Chameleon` largely improves AUT of different DL-based detectors compared to SOTA baselines.

Acknowledgment

This work was supported in part by the National Science and Technology Major Project under Grant 2025ZD0620000, in part by NSFC under Grant 62302452, Grant 62425201, Grant 62132011, in part by Zhejiang Provincial Natural Science Foundation of China under Grant LQ23F020019, and in part by the China Postdoctoral Science Foundation under Grant Number 2025M771583.

References

[1] Cloud storage. https://cloud.google.com/storage/?hl=en#pricing.
[2] Crowdsourcing. https://www.mturk.com/.
[3] Threat intelligence sharing. https://threatconnect.com/.
[4] Applying pca for traffic anomaly detection: Problems and solutions. In *Proc. of IEEE INFOCOM*, 2009.
[5] WIDE. Accessed January 2021. Mawi working group traffic archive. https://mawi.wide.ad.jp/mawi/.
[6] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. INSOMNIA: Towards concept-drift robustness in network intrusion detection. 2021.
[7] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.
[8] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *Proc. of IEEE S&P*, 2022.
[9] Abdullah Bin Jasni, Akiko Manada, and Kohei Watabe. Diffupac: Contextual mimicry in adversarial packets generation via diffusion model. *Advances in Neural Information Processing Systems*, 37:133846–133878, 2024.
[10] Christopher Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:531–537, 2006.
[11] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.
[12] Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *Proc. of USENIX Security*, pages 1127–1144, 2023.
[13] Euijin Choo, Mohamed Nabeel, Mashael Alsabah, Issa Khalil, Ting Yu, and Wei Wang. Devicewatch: a data-driven network analysis approach to identifying compromised mobile devices with graph-inference. *ACM Transactions on Privacy and Security*, 26(1):1–32, 2022.
[14] Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. Lifelong anomaly detection through unlearning. In *Proc. of ACM CCS*, 2019.
[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proc. of ACM CCS*, 2017.
[16] Yong Fang, Kai Li, Rongfeng Zheng, Shan Liao, and Yue Wang. A communication-channel-based method for detecting deeply camouflaged malicious traffic. *Computer Networks*, 197:108297, 2021.
[17] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proc. of ACM CCS*, 2021.
[18] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *Proc. of NDSS*, 2023.
[19] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.
[20] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.
[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Journal of Machine Learning Research*, 15:315–323, 2011.
[22] Dongqi Han, Zhiliang Wang, Wenqi Chen, Kai Wang, Rui Yu, Su Wang, Han Zhang, Zhihua Wang, Minghui Jin, Jiahai Yang, et al. Anomaly detection in the open world: Normality shift detection, explanation, and adaptation. In *Proc. of NDSS*, 2023.
[23] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C*, 1979.
[24] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[25] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proc. of ICLR*, 2016.

[26] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *Proc. of IEEE ICASSP*, volume 4, pages IV–317. IEEE, 2007.

[27] Wenjie Hu, Yihua Liao, and V Rao Vemuri. Robust anomaly detection using support vector machines. In *Proceedings of the international conference on machine learning*, pages 282–289. Citeseer University Park, PA, USA, 2003.

[28] Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. Label propagation for deep semi-supervised learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5070–5079, 2019.

[29] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. Kargus: a highly-scalable software-based intrusion detection system. In *Proc. of ACM CCS*, pages 317–328, 2012.

[30] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *Proc. of IEEE S&P*, 2020.

[31] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proc. of EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, 2016.

[32] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *Proc. of USENIX Security*, 2017.

[33] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D Joseph, and JD Tygar. Approaches to adversarial drift. In *Proc. of ACM workshop on Artificial intelligence and security*, pages 99–110, 2013.

[34] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proc. of the national academy of sciences*, 114(13):3521–3526, 2017.

[35] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM computer communication review*, 35(4):217–228, 2005.

[36] Adrian Shuai Li, Arun Iyengar, Ashish Kundu, and Elisa Bertino. Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples. In *Proc. of NDSS*, 2025.

[37] Peiyang Li, Ye Wang, Qi Li, Zhuotao Liu, Ke Xu, Ju Ren, Zhiying Liu, and Ruilin Lin. Learning from limited heterogeneous training data: Meta-learning for unsupervised zero-day web attack detection across web domains. In *Proc. of ACM SIGSAC CCS*, 2023.

[38] Wenhao Li, Xiao-Yu Zhang, Huaifeng Bao, Haichao Shi, and Qiang Wang. Prograph: Robust network traffic identification with graph propagation. *IEEE/ACM Transactions on Networking*, 2022.

[39] Junjie Liang, Wenbo Guo, Tongbo Luo, Honavar Vasant, Gang Wang, and Xinyu Xing. Fare: Enabling fine-grained attack categorization under low-quality labeled data. In *Proc. of NDSS*, 2021.

[40] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proc. of ACM Web Conference*, 2022.

[41] Anjin Liu, Yiliao Song, Guangquan Zhang, and Jie Lu. Regional concept drift detection and density synchronized drift adaptation. In *Proc. of IJCAI*, 2017.

[42] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[43] Yang Lu, Yiu-Ming Cheung, and Yuan Yan Tang. Adaptive chunk-based dynamic weighted majority for imbalanced data streams with concept drift. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2764–2778, 2019.

[44] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proc. of NDSS*, 2018.

[45] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proc. of ACM ICML*, pages 625–632, 2005.

[46] Cláudia Pascoal, M Rosario De Oliveira, Rui Valadas, Peter Filzmoser, Paulo Salvador, and António Pacheco. Robust feature selection and robust pca for internet traffic anomaly detection. In *Proc. of IEEE INFOCOM*, 2012.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proc. of USENIX Security*, pages 729–746, 2019.

[50] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[51] Viraj Prabhu, Shivam Khare, Deeksha Kartik, and Judy Hoffman. Sentry: Selective entropy optimization via committee consistency for unsupervised domain adaptation. In *Proc. of IEEE/CVF ICCV*, pages 8558–8567, 2021.

[52] Yuqi Qing, Qilei Yin, Xinhao Deng, Yihao Chen, Zhuotao Liu, Kun Sun, Ke Xu, Jia Zhang, and Qi Li. Low-quality training data only? a robust framework for detecting encrypted malicious network traffic. In *Proc. of NDSS*, 2024.

[53] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

[54] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.

[55] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2:169–194, 1998.

[56] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[57] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security*, 16:2367–2380, 2021.

[58] Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799–3821, 2007.

[59] Chris Sinclair, Lyn Pierce, and Sara Matzner. An application of machine learning to network intrusion detection. In *Proc. of IEEE ACSAC*, 1999.

[60] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. of IEEE S&P*, pages 305–316. IEEE, 2010.

[61] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Proc. of the first workshop on building analysis datasets and gathering experience returns for security*, pages 29–36, 2011.

[62] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei, Tao Wei, Yanfei Xu, et al. Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks. In *Proc. of IEEE INFOCOM*, 2020.

[63] Thijs Van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten Van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Proc. of NDSS*, 2020.

[64] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020.

[65] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proc. of ACM CCS*, 2015.

[66] Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, Dongqi Han, Han Zhang, Xingang Shi, and Jiahai Yang. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Transactions on Information Forensics and Security*, 17:3972–3987, 2022.

[67] Xian Wang. Enidrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift. In *Proc. of IEEE ACSAC*, 2022.

[68] Feng Wei, Hongda Li, Ziming Zhao, and Hongxin Hu. Xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *Proc. of USENIX Security*, 2023.

[69] Kun Xie, Xiaocan Li, Xin Wang, Jiannong Cao, Gaogang Xie, Jigang Wen, Dafang Zhang, and Zheng Qin. On-line anomaly detection with high accuracy. *IEEE/ACM transactions on networking*, 26(3):1222–1235, 2018.

[70] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. Cade: Detecting and explaining concept drift samples for security applications. In *Proc. of USENIX Security*, 2021.

[71] Shui Yu, Guofeng Zhao, Song Guo, Yang Xiang, and Athanasios V Vasilakos. Browsing behavior mimicking attacks on popular web sites for large botnets. In *Proc. of IEEE INFOCOM WKSHPS*, 2011.

[72] Chaoyun Zhang, Xavier Costa-Perez, and Paul Patras. Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms. *IEEE/ACM Transactions on Networking*, 30(3):1294–1311, 2022.

[73] Rongqian Zhang, Senlin Luo, Limin Pan, Jingwei Hao, and Ji Zhang. Generating adversarial examples via enhancing latent spatial features of benign traffic and preserving malicious functions. *Neurocomputing*, 490:413–430, 2022.

[74] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *Proc. of ACM CCS*, pages 757–770, 2020.

[75] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

APPENDIX

A. Statistics of Datasets

We show the statistics of evaluated datasets in Tab. IX. The ratios of stealthy traffic samples in the S-Nomal and S-Malicious datasets are 100% and 64.11%, respectively.

Table IX: Statistics of public and synthetic datasets.

Type	Dataset	#Malicious Samples	#Normal Samples	#Feature Dimension
Public Datasets	Kyoto 2006+	1,075,040	2,924,960	542
	CIC-IDS2017	431,159	869,063	78
	RWDIDS2022	579,482	1,470,791	200
Synthetic Datasets	S-Normal	109,389	240,611	542
	S-Malicious	127,421	222,579	78
	NS-Normal	100,000	300,000	200
	NS-Malicious	125,000	375,000	200

B. Implementation of Anti-Concept Drift Approaches

In this section, we describe the implementations of the anti-concept drift approaches used as baselines, i.e., CADE [70], HICL [12], TRANS [8], INSOMNIA [6] and OWAD [22]. We faithfully implement CADE[2], HICL[3], INSOMNIA[4] and OWAD[5] based on their public source codes and use the parameter tunning methods recommended in their papers or the default configurations to set their hyper-parameters.

For TRANS[6], we also use its public source codes and recommened hyper-parameter values. Further, we make several improvements to reduce its computational overhead. In particular, first we separate the Non-Conformity Measures (NCM) calculation process of training and test samples, thus reducing the time complexity from $O(mn)$ to $O(m+n)$, where m and n represent the number of training and testing samples, respectively. Second, we replace the for-loop operations with array-based parallel computation. Last, we unify the types of numerical values to eliminate the overhead of type conversions during the numerical computation process. Note that these improvements only reduce the computational overhead of TRANS, without affecting its anti-concept drift capability. We also use these methods to accelerate the representative sample selection process of our framework.

C. Malicious Traffic Detectors Implementation

We implement the MLP-based malicious traffic detector according to the design in the existing work [12]. In particular, it consists of three layers, including an input, a hidden, and an output layer. The hidden layer is with 128 units and uses the sigmoid activation function. The output layer utilizes the softmax function to output the binary classification result, i.e., normal or malicious. We employ the Adam optimizer with a learning rate of 10^{-4} for optimization. The batch size and the number of training epochs are 256 and 100, respectively.

We implement the unsupervised KitNet [44] detector based on its public source codes. The KitNet detector is built on an autoencoder architecture, which consists of a four-layer encoder and a four-layer decoder. The hidden layers in the encoder and decoder use the ReLU activation function. We also use the default hyper-parameter configurations of KitNet, e.g., the batch size is 1024, the number of training epochs is 20, and the learning rate is 10^{-4}.

We implement two other detectors using the sequence-based DL models. In particular, we implement a supervised detector based on the RNN model. The design of this detector and the hyper-parameter configurations are the same as the existing work [68]. It consists of a recurrent neural network layer with 64 hidden units and a fully connected layer with the sigmoid activation function. We also use the Adam optimizer with a learning rate of 10^{-4} for optimization. The batch size and the number of training epochs are 128 and 100, respectively. In addition, we implement an unsupervised detector based on the LSTM model [15]. It utilizes an LSTM layer with 64 hidden units and a fully connected layer. we set its hyper-

[2]CADE: https://github.com/whyisyoung/CADE
[3]HICL: https://github.com/wagner-group/active-learning
[4]INSOMNIA: https://s2lab.cs.ucl.ac.uk/projects/tesseract/
[5]OWAD: https://github.com/dongtsi/OWAD
[6]TRANS: https://s2lab.cs.ucl.ac.uk/projects/transcend/

parameters the same as KitNet [44] and its sequence window size to 10 [22].

We also implement two detectors using the graph-based DL models. Specifically, we implement a supervised detector based on the GIN model [57]. It consists of two graph convolution layers with 64 units and a fully connected layer, where each graph convolution layer utilizes a three-layer MLP model to extract features. The batch size and learning rate are set to 128 and 10^{-4}, respectively. Besides, we realize an unsupervised detector based on the GNN model [66]. It applies a graph-based autoencoder architecture, where the encoder contains two GCN (Graph Convolutional Network) layers and the decoder uses a fully connected layer. The hyper-parameters of this detector are the same as KitNet [44].

D. Other Hyper-parameter Configurations

We set the weight reduction coefficient A in the weight ω_t of each window as the reciprocal of the number of windows passed through. The μ in the label propagation algorithm is set to 0.6, which is the same as in existing studies [75], [28]. The maximum number of iterations of the multi-object optimization is set to 50 to reduce the time for drift detection. Besides, the coefficient B in the incremental training loss for unsupervised DL-based detectors is set to 5, which is consistent with the setting in UNLEARN [14].

E. Applicability to Traditional ML-based Detectors.

Some malicious traffic detectors [60], [46], [58], [35], [69], [4], [59] are powered by traditional Machine Learning (ML) algorithms, which also suffer from performance degradation under concept drift. Our framework is applicable to the ML models with explicit loss functions defined, i.e., computing the difference between the model's output and the target to optimize the model's parameters, e.g., SVM [24], [27]. The ML model's categorical outputs can be converted into probabilities via the probability calibration method [50], [45]. For the ML models without an explicit loss function, e.g., the Markov model [53], we can use our framework to infer the labels of testing traffic and retrain these models to maintain their performance under concept drift.