INSOMNIA: Towards Concept-Drift Robustness in Network Intrusion Detection

Anonymous Author(s)

ABSTRACT

Despite decades of research in network traffic analysis and incredible advances in artificial intelligence, network intrusion detection systems based on machine learning (ML) have yet to prove their worth. One core obstacle is the existence of concept drift, an issue for all adversary-facing security systems. Additionally, specific challenges set intrusion detection apart from other ML-based security tasks, such as malware detection.

In this work, we offer a new perspective on these challenges. We propose INSOMNIA, a semi-supervised intrusion detector which continuously updates the underlying ML model as network traffic characteristics are affected by concept drift. We use active learning to reduce latency in the model updates, label estimation to reduce labeling overhead, and apply explainable AI to better interpret how the model reacts to the shifting distribution.

To evaluate INSOMNIA, we extend TESSERACT—a framework originally proposed for performing sound time-aware evaluations of ML-based malware detectors—to the network intrusion domain. Our evaluation shows that accounting for drifting scenarios is vital for effective intrusion detection systems.

CCS CONCEPTS

• **General and reference** → **Evaluation**; • **Security and privacy** → **Network security**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Network Security; Machine Learning

ACM Reference Format:

Anonymous Author(s) . 2021. INSOMNIA: Towards Concept-Drift Robustness in Network Intrusion Detection. In *Proceedings of ACM Workshop on Artificial Intelligence and Security (Anonymous Submission to ACM AISec 2021).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In their landmark paper, Sommer and Paxson claimed that the reason machine learning (ML) had not yet been applied to intrusion detection—despite successes in other areas—was because *"[...] the intrusion detection domain exhibits particular characteristics that make the effective deployment of machine learning approaches fundamentally harder than in many other contexts."* [56].

In the decade since, with the widespread popularity of deep learning, the use of Deep Neural Networks (DNNs) has emerged as a valuable candidate for designing network intrusion detection systems (NIDS) [3–6, 15, 42, 54, 66]. However, despite a number of successes, many challenges still remain, and as our understanding of the area grows, yet more challenges arise beyond those originally outlined in Sommer and Paxson [56]. In this work, we tackle a set of open challenges which limit the practicality of current methods: the non-uniform distribution of network traffic over time, the high cost of labeling, the latency during model updates, and the lack of explainability (§2).

Core to all these challenges is the notion of *concept drift* [44]. Many previous methods typically follow the assumptions of traditional approaches: that the distribution of traffic data is stationary. Due to this, features which appear stable in the training data may appear adequate for describing *future* network flows. However, this i.i.d. assumption is invalid in modern network traffic environments where malicious activities are often polymorphic and continuously evolving as attackers adapt to defenses [25]. Due to this, new attacks or evasion strategies appear and it becomes difficult to distinguish between malicious and benign behavior [20, 32].

To illustrate this, we apply a baseline vanilla DNN (Appendix A) and Kitsune [42], a state-of-the-art NIDS based on an ensemble of autoencoders, to a recently revised version of the CICIDS2017 dataset [23]. These approaches assume i.i.d. data and do not include a mechanism to mitigate the impact of drift. The two approaches identify almost *zero attacks* across the 3 days of test data (Table 1), clearly demonstrating that a modern NIDS must be proactive in the face of concept drift. Ultimately, NIDS should *adapt to changes* in the distribution of network traffic characteristics [37]. However, designing an effective adaptation mechanism is nontrivial and requires innovations to feature learning, inference, and the ongoing operational deployment of the system.

We believe deep learning-based approaches are a promising starting point for developing NIDS that are robust to drift, since the nonlinear activation functions of DNNs may allow models to maintain their accuracy under drifting conditions, as originally evidenced by Pendlebury et al. [48]. Additionally, DNNs are responsive to incremental learning, which allows them to adapt to the new data distribution without the need to restart training from the entirety of accumulated traces [16].

In this paper, we propose INSOMNIA: a framework that addresses challenges to network intrusion detection in the presence of drift. INSOMNIA uses a DNN as its central underlying classifier and to reduce the impact of latency introduced by model updates, we use active learning [50] to update using only those points that would maximize information gain. To avoid the high overhead of labeling with a human oracle, INSOMNIA is semi-supervised, using a Nearest Centroid Neighbor classifier (NC) [17] to estimate labels for the selected points (thus avoiding manual labeling in updates). Finally, to understand how drift manifests in the dataset, we use a

Anonymous Submission to ACM AISec 2021, Due November 13th, 2021, Seoul, South Korea

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

permutation-based variable-importance measure [14, 24] to provide global model explanations over time.

To evaluate INSOMNIA, we extend TESSERACT [48], a framework originally proposed to remove experimental bias and perform time-aware evaluations in the malware detection domain. One key aspect of TESSERACT is that it ensures temporally consistent dataset splits, with the training data preceding the test data, and the test data partitioned into consecutive time periods of equal size (e.g., one month). To capture the performance of a classifier over time, it uses the Area Under Time (AUT) metric.

Applying TESSERACT to the network traffic domain is not immediate. Firstly, the time granularity at which attacks occur in networks is much smaller, possibly in the order of minutes or hours. Secondly, the data is not uniform over time, e.g., there is typically less activity at weekends or at night, and there may be quiet periods where no attacks occur at all. Thirdly, operating in such short timeframes has implications on retraining, both in terms of processing time and also in terms of labeling capacity, which should be taken into account during the evaluation.

In summary, this paper makes the following contributions:

- We outline a set of open challenges facing modern ML-based intrusion detectors (§2) and propose INSOMNIA, a semi-supervised approach that builds on active learning, label estimation, and explainable AI in order to tackle them (§3).
- We extend TESSERACT [48] to the network intrusion domain and use it to evaluate INSOMNIA on a recently revised version of the CICIDS2017 dataset [23]. Our results demonstrate the need to proactively address changing distributions in network traffic as INSOMNIA significantly outperforms baselines that do not account for drift (§4).

Our study highlights current challenges and promising trends in effectively addressing concept drift for network intrusion detection with deep learning, while providing explanations that are fundamental for understanding findings and the behavior of models. To foster future research, we release our code to the community (§7).

2 CHALLENGES AND MOTIVATION

We outline a number of remaining open challenges which limit the practicality of existing solutions [e.g., 5, 42] in drifting settings.

Non-uniformity of data distribution over time. Crucially, NIDS must handle the lack of uniformity in the distribution of malicious traffic traces. Attacks of different types can start and stop abruptly as different adversaries launch different engagements at different phases of the attack lifecycle [38]. This means that not all categories of malicious behavior are represented uniformly across the training data which is problematic for models that update incrementally or in an online fashion. Note that this is in contrast to malware detection where, although new attacks do appear over time, successful families tend to be ubiquitous [58].

Additionally, concept drift can even affect traffic features which are seemingly well-established. A common case is when benign behavior, contrary to malicious behavior, exhibits a very gradual drift as user habits change. In these cases it is not necessary to relearn from the entirety of network traffic traces, but identifying which characteristics change and then tuning the NIDS to traces which exhibit such changes.

DNN models should accommodate the appearance of new categories of behavior without losing inference capability on older categories. A possible solution is to train a base DNN model on historical labeled data and then update it over time to fit unlabeled incoming traces via transfer learning [49]. In the presence of zero-day attacks, the past model may be structurally extended to incorporate new model branches [49].

Cost of labeling. Very accurate intrusion detection models are commonly trained using supervised learning by processing large amounts of labeled traces [5, 36, 66]. However, manually labeling network traffic is costly and, due to concept drift, models need to be updated frequently with freshly labeled examples to maintain continuously high accuracy over time.

Active learning query strategies [2] investigate how to select a subset of new samples that, if manually labeled and incorporated into the training set, would add the most information to the model, and are therefore the most valuable samples to label. A common active learning query strategy is *uncertainty sampling* [35] which selects the most "uncertain" classifications (e.g., those closest to the decision boundary) for manual labeling and retraining. The intuition is that these are the most relevant for readjusting decision boundaries, which can become blurred by drifting examples.

Pendlebury et al. [48] apply active learning with uncertainty sampling to malware detection in which a human oracle is queried each 'month' for manual labels with which the model is updated. They follow estimates by Miller et al. [41] that an average company could manually label 80 applications per day. In such a setting, new malware variants take time to be developed and, for mobile malware at least, there is a probation period where apps are vetted before appearing in a marketplace. This means that the update operation can be scheduled at longer intervals. However, for intrusion detection systems, network traffic is highly diverse and can fluctuate suddenly, requiring faster response times which can put strain on labeling capacity [19].

Additionally, network traffic traces may be acquired irregularly over time which can compromise the effectiveness of scheduling model updates at regular intervals. If very little traffic occurs, computation is wasted by updating the model with scant new information. If a large amount of traffic occurs, new attack categories may be processed but not detected until they reappear in the next interval. Such an occurrence can cause a sharp decrease in performance. Alternatively, we propose using *count-based windows* to update the model once a sufficient number of traces have been acquired (see Figure 2). Furthermore, to reduce the cost of using human oracles, manual annotators may be replaced or supplemented with automated annotation mechanisms that provide regular feedback at a very limited cost [65], a strategy we explore in this work (§3).

Update latency. A further issue is the latency induced when updating an intrusion detection model [39]. In principle, the new model should be available in near-real time, i.e., before the subsequent traffic trace arrives for classification. In practice, the new model is only available after some delay, with the previous iteration of the model being used on traces that appear in the meantime.

One solution to reducing the learning latency is the use of transfer learning [46] in which a model trained to solve one task is applied to another. In particular we can adopt *fine-tuning*, a simple

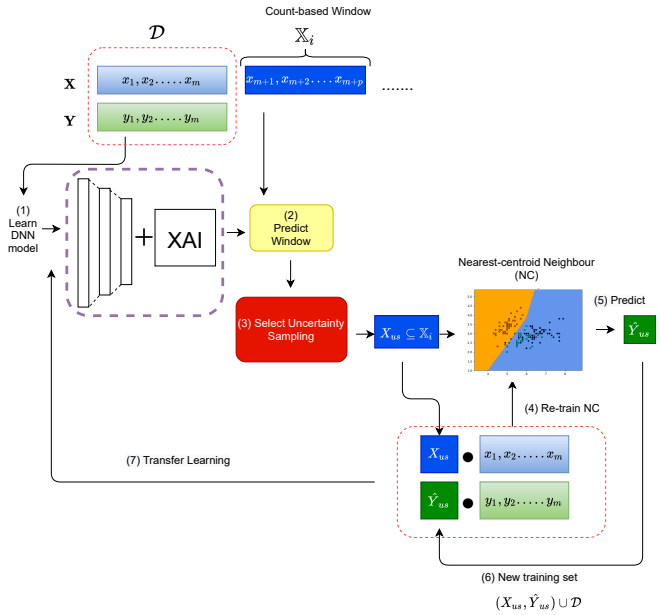


Figure 1: A block diagram showing the different components of the INSOMNIA framework

application of transfer learning in deep learning [59]. In fine-tuning, a model is trained on data from the target distribution, but rather than the weights being randomly initialized, they are pretrained on data from a different—but related—distribution. In the NIDS context, this allows the model to adapt to the drifting distribution without retraining from scratch, which would incur significant overhead.

Explainability. Although deep learning techniques have been widely used to obtain superhuman classification capabilities, the trained models are in most cases *black-box* models [31].

Models used for intrusion detection are no different, and are implicitly represented in numerical form as synaptic weights of the network. It is generally difficult, if not impossible, to interpret these weights without further tool support. However, the interpretability of intrusion detection systems is of fundamental importance to understanding the decision of the model and informing downstream actions on how to prevent evolving attacks.

Currently, the machine learning community is dedicating increasing efforts towards developing eXplainable Artificial Intelligence (XAI) techniques for interpreting deep learning models [63]. A recent study by Warnecke et al. [62] applied explainability methods to provide explanations for DNN decisions in malware detection and vulnerability discovery. In this work we consider applying XAI to NIDS *in a temporal setting*, to explain how the black box is changing over time to fit to new attack categories.

3 METHODOLOGY

In this section we present INSOMNIA (Incremental training iNtrusion System Over tiMe-stamped Network traffIc dAta), a semi-supervised methodology that combines incremental, active, and transfer learning to overcome the challenges described in §2. Additionally, it applies XAI to provide post-hoc explanations of how the model changes over time to fit the appearance of new attack categories in the network traffic.

3.1 Overview

INSOMNIA initially learns an intrusion detection model from a collection of labeled historical network traffic traces. It then continues in an unsupervised manner, monitoring incoming unlabeled traces, and adapting the model over time to fit the drifting conditions of the network. The updates are facilitated by a learnt oracle mechanism which produces class estimates (i.e., pseudo-labels) of new traces. A block diagram of INSOMNIA is reported in Figure 1.

INSOMNIA assumes that only a limited quantity of labeled traces are available initially (acquired during a data collection stage) with an abundance of unlabeled traces later acquired over time (when new network traffic traces arrive). Formally, the input is an ordered multiset $\mathcal{S}$ of time-stamped network traffic traces,

$$\mathcal{S} = \{\!\{ (\boldsymbol{x}_t, y_t) : t = 1, 2, \ldots, m, \ldots \}\!\},$$

where $\boldsymbol{x}_t$ is a vector of flow-level traffic feature values and y_t is the corresponding binary label denoting a *benign* trace or an *attack* trace. We assume that y_t is known where $1 \leq t \leq m$, while y_t is unknown where $t > m$.

INSOMNIA operates in three phases: an initialization phase, an incremental learning phase, and an explanation phase.

In the initialization phase, the labeled traces are used to train the intrusion detection model. Additionally, we train a distinct oracle mechanism to estimate the true labels of the incoming examples. This mechanism replaces the human oracle or external information source commonly used in active learning [2, 7]. The two models are distinct to ensure the predictive model is not affected by negative feedback loops caused by training with its own predictions.

In the incremental learning phase, new unlabeled traces are consumed consecutively and processed in batches of equal size. The intrusion detection model and the label estimator are continuously updated with the new traces which are unlabeled at inference time and class-estimated before the model update.

Finally, in the explanation phase, the global relevance of features involved in the intrusion detection model decisions is monitored over time to explain how the model has updated to fit the drifting characteristics of the network traffic. In the following subsections we describe these three phases in more detail.

3.2 Initialization Phase

In this phase, we consider the initial m labeled traces that form the training set $\mathcal{D} \subset \mathcal{S}$. We process $\mathcal{D}$ to train both the intrusion detection and oracle models. As an intrusion detection model, we train a DNN model. As the label estimator, we train a Nearest Centroid Neighbour classifier (NC) [60].

NC is an efficient classification algorithm that assigns a sample to the same class as the training examples whose mean (*centroid*) is closest to the new sample. We note that NC relies on a metric learning strategy that takes advantage of the proximity between samples to highlight hidden patterns useful for intrusion detection. This learning strategy is in stark contrast to that of DNNs. The use of NC for label estimation follows the *cluster assumption* of semi-supervised learning, i.e., that points tend to form discrete clusters, and that points in the same cluster are more likely to share a label than those that are not [28].

The two diverse learners, DNN and NC, provide complementary predictive capabilities and we hypothesize that the NC can act as an independent oracle labeling mechanism to better label traces the DNN predicts with uncertainty as a form of *co-training* [13]. In §4.3.3, we empirically verify the effectiveness of this hypothesis.

3.3 Incremental Learning Phase

After initialisation, the DNN model classifies new unlabeled traces as either *benign* or *attack* instances. As they are classified, new traces are aggregated one-by-one into batches of size p. Once a batch $\mathbb{X}_i$ is complete, the incremental learning operation is triggered to update the DNN model. This operation consists of three steps:

(1) An uncertainty set $X_{\mathcal{US}}$ is constructed by selecting the traces of $\mathbb{X}_i$ which are assigned the least certain predictions by the DNN model;
(2) Next, the training set $\mathcal{D}$ is augmented with traces of $X_{\mathcal{US}}$ pseudo-labeled using the NC-based oracle;
(3) $\mathcal{D}$ is processed to update the DNN model, as well as the NC-based oracle.

To construct $X_{\mathcal{US}}$ we adopt the *uncertainty sampling* (US) query strategy [48] to select points with the least certain predictions for which labeling will provide the most new information. Specifically, we determine the uncertainty of DNN classifications by considering outputs of the DNN's softmax layer. The lower the softmax confidence value, the more uncertain the classification is. We select the top σ% most uncertain traces of $\mathbb{X}_i$ to form $X_{\mathcal{US}}$. The trace selection rate σ is a user-defined parameter.

Next, traces in $X_{\mathcal{US}}$ are labeled with the current NC-based oracle and added to $\mathcal{D}$, which is used to update both the DNN model and the NC-based oracle. The DNN model is updated with the fine-tuning operation that adapts the weights of the previously trained DNN model to the potentially drifting distribution of $\mathcal{D}$.

The class centroids of the NC-based oracle are recomputed on $\mathcal{D}$. This is a form of *self-learning* [67], in which a classification algorithm learns from a labeled dataset that is augmented with new examples labeled by the classifier itself. By retaining the initial training examples that have ground truth labels, we mitigate the potential catastrophic effects introduced by low quality pseudo-labels. This also ensures new traces are not inordinately more relevant than older traces, to avoid unlearning previous attack behavior.

Note that the update operation introduces a learning latency until the new DNN is ready. Therefore, in the interim, the old DNN model (i.e., the model being updated) must be used to classify the incoming traces, which may induce a temporary performance decay relative to the amount of drift.

3.4 Explanations Phase

Finally, INSOMNIA relies on XAI to provide post-hoc explanations of how the DNN model adjusts over time to fit the appearance of new attack categories in the network traffic.

We use the moDel Agnostic Language for Exploration and eXplanation (DALEX) [10], a framework for explaining predictive black-box models. DALEX implements techniques for understanding both the global and local structure of a black-box model. In INSOMNIA, we integrate the global explanation methodology, which allows us to explain the behavior of the DNN by measuring the global relevance of different features (i.e., observed traffic characteristics).

DALEX uses a permutation-based variable-importance measure to quantify the relevance of each feature [14, 24]. For each feature, its effect is removed by resampling or permuting the values of the feature and a loss function compares the performance before and after. Intuitively, if a feature is important, randomly permuting its values will cause the loss to increase.

By inspecting how the feature importance changes over time, we can identify those that remain relevant, as well as those that are redundant. Importantly, by analyzing what features increase in relevance as new attack categories appear, we can identify the features that contribute the most to the characterization (and detection) of each attack category.

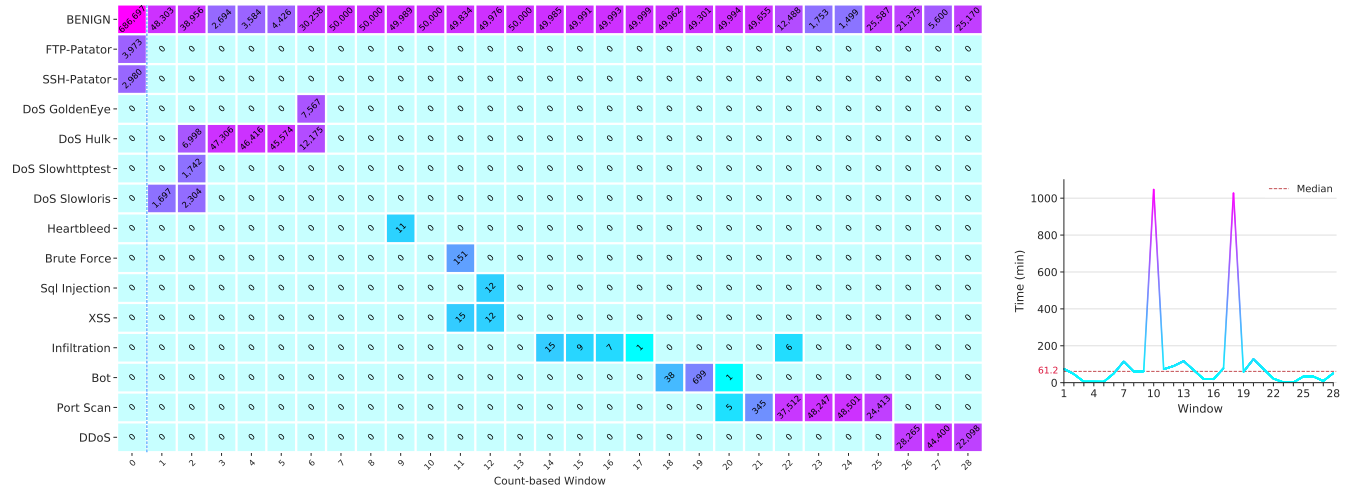
4 EVALUATION

In this section we perform a time-aware evaluation of the detection and explaination capabilities of INSOMNIA[1] on CICIDS2017, a recently revised network intrusion dataset [23, 52]. To this end, we extend the TESSERACT [48] framework to support time-aware evaluation of network intrusion detection tasks. In particular, we add functionality to temporally partition data using count-based windows rather than time splits used in the original work, as well as to compute the impact of latency during model updates.

4.1 CICIDS2017 Network Traffic Data

The CICIDS2017 dataset [52] from the Canadian Institute for Cybersecurity provides data collection of a complete network testbed of different devices and operating systems, in which separate victim

[1] See Appendix A for DNN architecture, implementation, and hyperparameter tuning.



(a) Category distribution

(b) Window length in minutes

Figure 2: CICIDS2017 overview. Figure 2a shows the overall distribution of traces per attack category for initialization and incremental phases. The initialization phase covers days 1 and 2 for a total of $m = 693,650$ labeled traces (block labeled 0). The incremental phase covers days 1, 2, and 3 spanning consecutive windows of $p = 50,000$ traces (blocks 1-28). Figure 2b shows the total time spanned by count-based windows (in minutes) for the incremental phase.

and attacker networks communicate over the Internet. Communication covers common network protocols, including HTTP, HTTPS, FTP, SSH, and SMTP. Profiling agents, trained beforehand on network events generated through genuine human interactions on the network, were used to generate realistic benign traffic, while attacks were identified according to a 2016 McAfee report [40]. Attacks include brute force attacks, Heartbleed, botnet communication, several variants of DoS and DDoS, infiltration, and web-related threats. In total, the dataset contains more than 2.8 million time-stamped traffic traces spanning 5 days and divided into 15 attack categories. Individual traffic traces are represented by a feature vector of 79 high-level statistical characteristics, manually engineered by intrusion detection experts and extracted using the tool CICFlowMeter [21, 34].

In this work, we use a refined version of CICIDS2017 recently released by Engelen et al. [23]. This version addresses flaws affecting the original dataset related to traffic generation, flow construction, feature extraction, and labeling that severely undermine its correctness, validity, and overall utility. The new version removes meaningless artefacts, dataset errors, and mislabeled traces, retaining 2,524,767 traces and 72 features. As expected in real deployments, the benign class is the majority class with 2,090,918 traces (82%) labeled as benign and 433,849 traces (18%) as attacks. However, we remark that attacks will be even less prevalent in real data, so it is necessary to take into account the base-rate fallacy [9] when interpreting the results.

4.2 Experimental Setup

We consider the first two days of the dataset as the labeled training set processed during the initialization phase of INSOMNIA (§3.2). The remaining three days are used as the test data, processed as unlabeled traces during the incremental learning phase (§3.3).

Figure 2a visualizes the distribution of traces across the 15 attack categories. We note that unseen attacks, appearing over time and causing concept drift, are those handled during the incremental learning phase (window 1 onwards). Figure 2b shows the length of each count-based window in minutes, with a median length of 61.2 minutes. The window size $p = |\mathbb{X}_i|$ is set to 50,000, while the initial training set size $m = |\mathcal{D}|$ is set to 693,650 to cover all traces in days 1 and 2. For the uncertainty sampling, we investigate the effects of varying the trace *selection rate* σ, with values of 20%, 50%, and 70%.

We evaluate the detection performance of the DNN model in terms of average F_1 and AUT(F_1).[2] F_1 is the harmonic mean of precision and recall, where precision ($TP/(TP+FP)$) measures the proportion of correct positive predictions and recall ($TP/(TP+FN)$) measures the model's ability to detect all attacks. AUT(F_1) is the area under the curve of F_1 computed over time [48] and is a time-aware metric specifically designed to capture the time decay of a security classifier.

We consider two scenarios; an ideal scenario where there is no learning latency during the incremental updates, and a more realistic scenario where this latency is taken into account. F_1 and AUT(F_1) are measured assuming the ideal scenario, without learning latency, where the DNN is fine-tuned at the end of $\mathbb{X}_i$ to classify all traces in $\mathbb{X}_{i+1}$. In the realistic scenario, new incoming traces from the subsequent window $\mathbb{X}_{i+1}$ will already have been encountered before the fine-tuning procedure of the DNN model has completed for window $\mathbb{X}_i$. To account for this delay, we present results for F_1

[2] More precisely, we use AUT(F_1, $28b_{50k}$), where b_{50k} denotes count-based blocks of 50,000 samples—in contrast to the original time-based notation where $12m$ would denote a year of month-sized periods, for example. For brevity we use the shorthand AUT(F_1) throughout.

Table 1: Performance and total TIME of No-Update, Kitsune [42], US+Oracle, and INSOMNIA (with NC) for different σ at 20%, 50%, and 70%.

Selected	Method	F_1 (%)	AUT(F_1)	F_1-D(%)	AUT(F_1)-D	TIME (min)
–	No-Update	0.0019	0.035	0.0019	0.035	–
	Kitsune [42]	0.0113	0.009	0.0113	0.009	–
20%	US+Oracle	74.57	32.73	74.40	32.61	445.24
	INSOMNIA	69.83	41.64	69.82	41.56	262.25
50%	US+Oracle	81.62	36.10	81.46	35.99	517.92
	INSOMNIA	80.88	42.39	80.40	42.17	428.39
70%	US+Oracle	90.85	44.99	90.74	44.68	587.96
	INSOMNIA	64.90	29.10	64.90	29.09	502.41

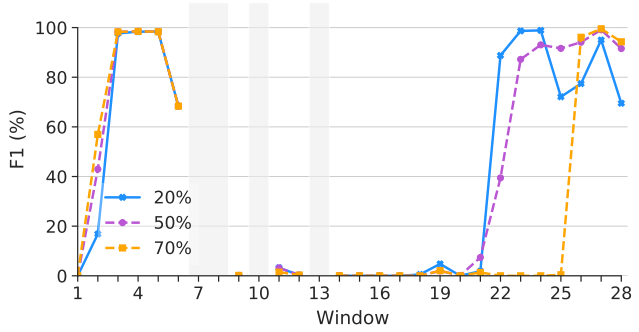


Figure 3: F_1 (%) of INSOMNIA computed on all traces across the consecutive windows at varying σ of 20%, 50%, and 70%. The grey spans correspond to windows in which no attacks occurred and for which F_1 is undefined.

and AUT(F_1) where the previous model is used up until the new model is ready, denoted as F_1-D and AUT(F_1)-D respectively.

Finally, we evaluate the efficiency of INSOMNIA by measuring the runtime taken to process all windows $\mathbb{X}_i$ (denoted as TIME). All experiments were executed on a Linux machine equipped with an Intel(R) Core(TM) i7-9700F CPU @ 3.00GHz and memory RAM of 32GB using a single GeForce RTX 2080.

4.3 Results

We report results considering several axes of performance: correctness of the predictions, quality of the uncertainty sampling strategy, accuracy of the label estimator, and utility of the explanations.

4.3.1 INSOMNIA Performance. Table 1 displays the results of our experiments comparing INSOMNIA against baselines with either no update, or a perfect update (using ground truth labels). For the uncertainty sampling query strategy, we compare results for different values of selection rate σ: 20%, 50%, and 70%. These results lead to several conclusions detailed below.

Comparison to baselines. We first compare INSOMNIA against three baselines: Kitsune [42], and two variants of INSOMNIA: No-Update and US+Oracle. No-Update does not implement any incremental learning phase, instead it reuses the model learned during the initialization phase to classify all subsequent traces. US+Oracle implements the incremental learning phase using the uncertainty sampling strategy, but relies on a human oracle to produce ground truth labels in place of the NC-based oracle. This represents an ideal upper bound on performance.

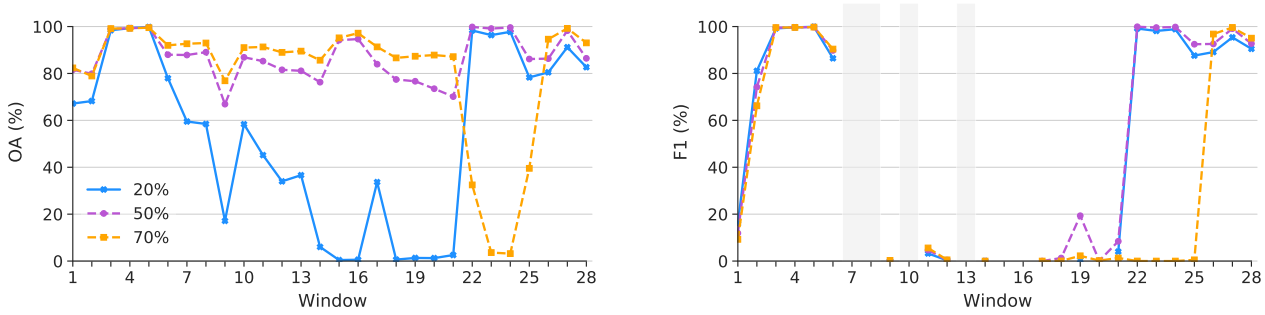
No-Update and Kitsune [42] both suffer from extreme performance decay, completely failing to recognise attacks that arise over time and were not encountered during initialization. While we note that Kitsune was not designed to withstand concept drift (and rather focuses on orthogonal research problems such as constrained resources), these results give empirical support to the idea that adaptation strategies such as incremental learning are important for addressing concept drift in network intrusion data.

As expected, INSOMNIA does not outperform US+Oracle, which uses ground truth labels rather than class estimates. However, the experimental configuration with $\sigma = 50\%$ has average F_1 very close to US+Oracle and even outperforms it in terms of the temporal AUT(F_1) metric. Significantly, the performance of INSOMNIA is much closer to that of US+Oracle than No-Update. This demonstrates that active learning with pseudo-labels can closely approximate performance with ground truth labels in scenarios where labeling capacity is limited.

Impact of selection rate. Next we analyze the effect of the selection rate σ on the performance of INSOMNIA. The highest F_1 is achieved with $\sigma = 50\%$, followed by 20%, while $\sigma = 70\%$ produces the lowest performance.

This performance comparison is further depicted in Figure 3 which shows F_1-D of INSOMNIA over consecutive windows for the three different σ. Notably, $\sigma = 70\%$ produces significantly worse performance for windows 22–25. To explore the cause of this behavior, we analyze the accuracy of the pseudo-labels generated by the NC-based oracle. Figures 4a and 4b compare the overall accuracy (OA) and F_1 of the label estimates computed for the 20%, 50%, and 70% selected uncertain traces. From this we observe that the low F_1 of the label estimator during windows 21–24 correlates to the poor performance of the DNN during the same timeframe, where $\sigma = 70\%$.

While the low F_1 of the label estimator in the preceding periods seems to have little detrimental effect on the performance, this is likely due to the extremely low prevalence of attacks during



(a) Overall accuracy of NC for selected points

(b) F_1 of NC for selected points

Figure 4: OA(%) and F_1 (%) of NC label estimations computed over the consecutive windows generated for the uncertain traces selected with the US strategy. Metrics are collected by varying σ among 20%, 50% and, 70%. The grey spans correspond to windows in which no attack traces where selected by the strategy and for which F_1 is undefined.

these windows compared to the large influx of port scanning traffic received in window 22. We reason that port scans may be naturally harder to distinguish from benign traffic, but that the large volume of port scans exacerbates the self-poisoning effect. Further analysis of the label estimator accuracy is included in §4.3.3.

Learning latency and runtime. We note a slight decrease in performance across all configurations when the learning latency is taken into account (e.g., F_1-D vs. F_1). This highlights how the update mechanism of INSOMNIA is sufficiently quick to avoid long latency periods, and that the system is robust to the onset of these periods when they do occur. We can conclude that latency does not threaten the ideal performance of INSOMNIA.

Similarly when considering runtime, we see that the total time taken to complete updates (TIME) positively correlates with the higher σ, increasing the risk of latency periods. This behavior is observed equally in the ideal scenario using a human oracle (US+Oracle) as with our NC-based estimator. However we note that our runtime calculations also assume that human-derived labels are available as soon as the model needs them; realistically, manually analyzing attack traffic can be very time intensive, which would further increase the runtime of US+Oracle.

Figure 5 depicts the time spent per window on completing the incremental learning operation for different values of σ. Windows marked with a red cross depict latency periods for which the model for that window is fine-tuned only after all traces from that window have already been classified (i.e., the new model is 'too late'). As expected, the total time and number of such periods increases with σ. On the other hand, as illustrated earlier, this does not necessarily suggest a trade-off between performance and latency with respect to the selection rate: high values of σ may cause catastrophic feedback loops due to the addition of low quality pseudo-labels.

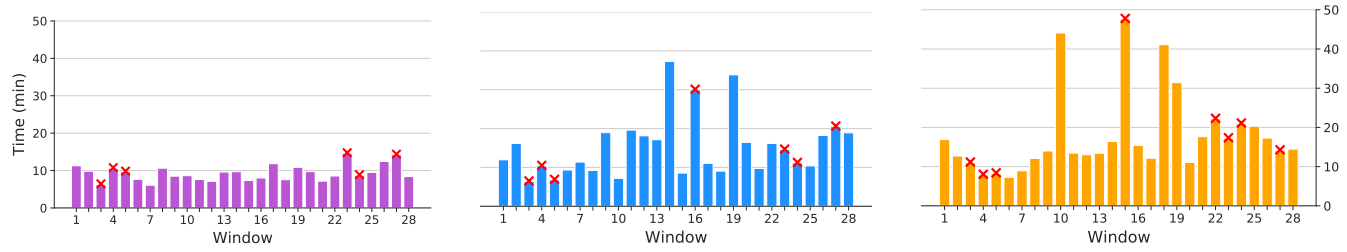
4.3.2 US Strategy Performance. We also perform an ablation study to inspect the effectiveness of the US strategy. We compare the accuracy of INSOMNIA to that of a variant (RS+NC) for which the query strategy has been replaced with the Random Sampling (RS) strategy [51]. The RS strategy randomly chooses traces to query with the NC-based oracle. We perform the comparison at $\sigma = 50\%$.

The results depicted in Figure 6 show that the performance of INSOMNIA with US is superior to the RS strategy, independent of the metric considered. This suggests that selecting traces with the most uncertain predictions does indeed result in a more efficient update that increases the information gained by the model.

4.3.3 Oracle Performance. We also analyse how well the NC classifier supports the DNN as an oracle. To this end, we compare the OA and F_1 per window on the traces selected with the US strategy at $\sigma = 50\%$ as classified by the DNN model versus the NC classifier.

Figure 7 shows the NC-based oracle can effectively improve the accuracy of the most uncertain classifications yielded by the DNN model (e.g., in windows 6–14 and 22–24). This shows that combining two learners with differing underlying concepts helps support high-quality pseudo-labels which can preserve the robustness of the DNN model over time. This avoids potential negative feedback loops, which could cause self-poisoning if the DNN updated itself using its own predictions. We emphasize that the relationship between the NC and the DNN are symbiotic—the NC relies on the DNN to provide the most uncertain predictions and the NC supplies estimates for corrected labels. The NC would not be able to operate independently in the same context unless it was supported by a human oracle to provide it with labels (or an alternative label estimator) to mitigate uncertain predictions.

4.3.4 Drift Explanation. Finally, we focus on explaining how the DNN model changes over time as new attack categories appear. For this analysis, we consider the DNN model learned with INSOMNIA at $\sigma = 50\%$ and we use DALEX to measure the global relevance of features based on the predictive capability of the DNN for each window. A significant change in the relevance of one or more features illustrates the DNN adapting to concept drift. The heatmap in Figure 8 depicts the ranked feature relevance for the top ten most relevant features per window. We note that the features *Idle Max* and *Protocol* are consistently ranked in the top ten positions (darkest cells) over time, while others gain relevance only with the appearance of different attack categories (cf. Figure 2a). Next we examine in depth how feature relevance changes for two sequence of windows: windows 0–2 and windows 22–23.



(a) $\sigma = 20\%$ **(b)** $\sigma = 50\%$ **(c)** $\sigma = 70\%$

Figure 5: Computation time (in minutes) spent completing the incremental learning phase of INSOMNIA for varying σ at 20%, 50%, and 70%. Crosses ($\times$) indicate latency periods for which the fine-tuning of the DNN model completes *after* all traces in the current window have been acquired.

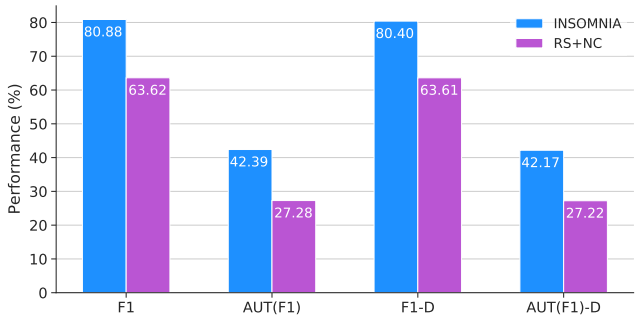


Figure 6: F_1 (%), AUT(F_1), F_1-D(%) and AUT(F_1)-D measured per window with both INSOMNIA and RS+NC with 50% of traces selected with the US strategy and the RS strategy, respectively.

Windows 0–2. As shown in Figure 2a, the model is initially trained on window 0 which contains only benign traces and attacks by password bruteforcers: *FTP–Patator* and *SSH-Patator*. Both *FTP-Patator* and *SSH-Patator* attacks disappear in subsequent windows, while denial-of-service (DoS) attacks appear with *DoS Slowloris* in windows 1 and 2, and *DoS Slowhttptest* in window 2.

Figures 9a–9c show the top ten globally relevant values identified with DALEX for the DNN initialized on window 0, fine-tuned on window 1, and fine-tuned on window 2, respectively. We note that there is a significant change in the relevance of several features from window 0 to window 1 as INSOMNIA updates the DNN model to detect the new *DoS Slowloris* traces.

Slowloris is a DoS attack that relies on creating several partial HTTP requests, keeping such connections open for as long as possible. Among relevant network features, one would then expect to observe the "maximum time a flow was idle before becoming active" (*Idle Max*)[3] to gain importance, as Figures 9a–9c depict. A similar reasoning can be applied to the other three new traffic features (*Protocol*, *Destination Port*, and *Bwd IAT Std*) that gain relevance for recognizing these attack traces. *Bwd IAT Total*, which was already identified as relevant in recognizing the bruteforce attacks, increases in relevance too. Similarly, some features become less relevant (i.e., *Idle Std*, *Bwd IAT Min*, *Bwd IAT Mean*, and *Fwd IAT Total*) while others drop from the top features entirely (i.e., *Fwd IAT Min* and *Init Win bytes backward*).

Three new features (*Min Packet Length*, *Flow IAT Min*, and *Packet Length Variance*) become relevant in window 2 as the model adapts to the change in DoS strategy (from *Slowloris* to *Slowhttptest*). The top-four features (*Idle Max*, *Idle Std*, *Protocol*, and *Bwd IAT Min*) do not change their relevance, two of which (*Idle Max* and *Protocol*) became more relevant in window 1. This suggests these features help the model stay stable in recognizing the *DoS Slowloris* attack as it persists into window 2.

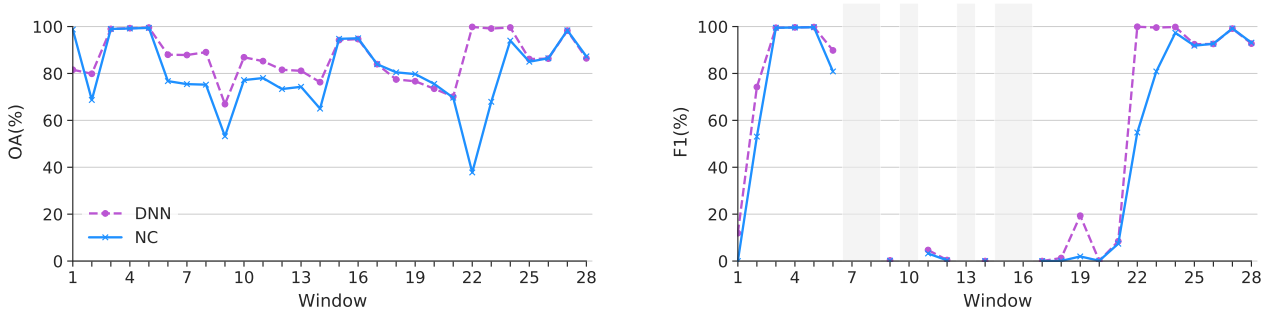
Windows 22-23. Figure 2a shows port scanning activity (*Port Scan*) appearing in a small number of traces at windows 20 and 21 (5 and 345, respectively), then significantly increasing in volume during windows 22 and 23 (37,512 and 48,501, respectively).

Figures 9d-9f show the top ten globally relevant values identified with DALEX for the DNN fine-tuned on windows 22, 23, and 24. We note that the feature *Bwd Packet Length Min*, which gains relevance in window 22 (as the port scanning activity increases), retains its relevance on windows 23 and 24, as port scanning activity is sustained. A similar trend is observed for *Idle Mean* that becomes relevant in window 22 and maintains its relevance throughout windows 23 and 24.

5 LIMITATIONS AND FUTURE WORK

The results of our evaluation highlight yet another important open problem for NIDS: detection of stealthy, low-prevalence attacks. INSOMNIA struggles to detect the few instances of the *Infiltration* attack at windows 14–17 and we observe that maintaining sensitivity to attacks with a very low base rate in the presence of high volume attacks such as DoS is very challenging—as is generalizing to attack categories of greatly different character. While generalizing across attack types remains a holy grail, future work may consider ensembles that tackle different attack types with separate models. Additionally, INSOMNIA's update mechanism does not have the opportunity to make use of knowledge learned from attacks which occur wholly in a single window (e.g., *Brute Force* at window 11). Reducing the window size may help with this, however it would also reduce the statistical support available each update.

[3] A comprehensive mapping of feature names to their corresponding description is available at https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt.



(a) Overall accuracy for selected points ($\sigma = 50\%$)

(b) F_1 for selected points ($\sigma = 50\%$)

Figure 7: OA(%) and F_1 (%) of the NC-based oracle versus the DNN model. Metrics are computed on the 50% of traces selected per window with the US strategy. The grey spans correspond to windows in which no attack traces where selected by the strategy and for which F_1 is undefined.

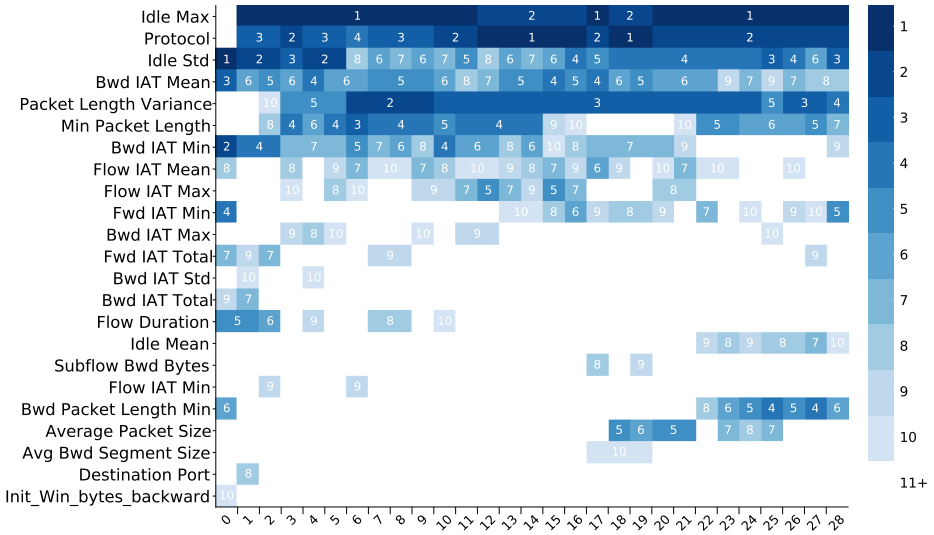


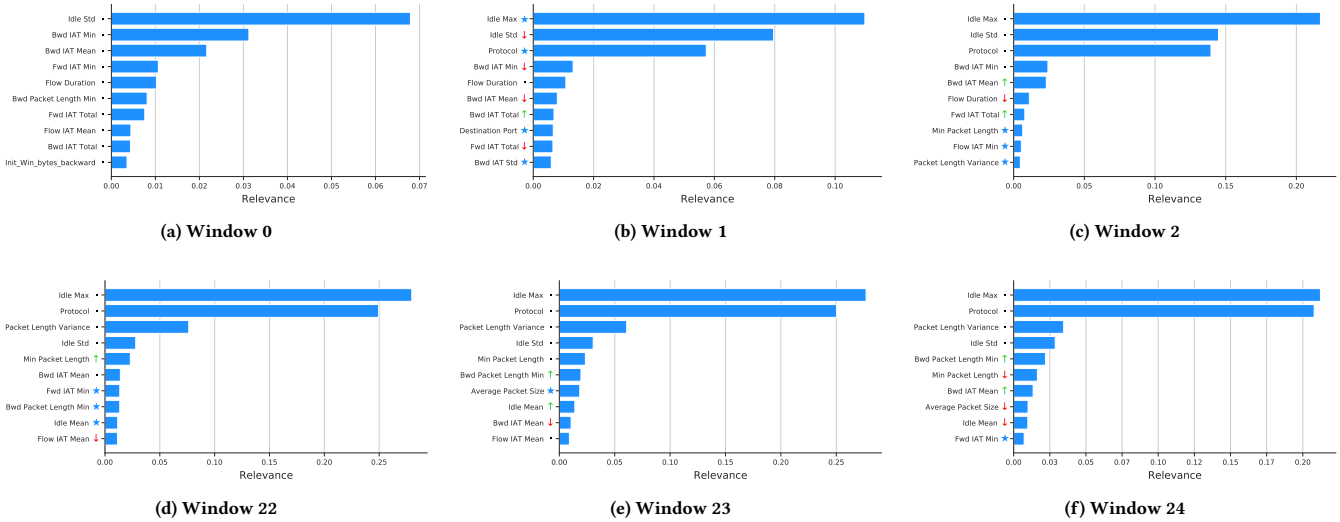
Figure 8: Feature ranking map of the DNN model learned with INSOMNIA at $\sigma = 50\%$. We plot the rank (1–10) of the features (axis Y), which are ranked in the top ten positions of the feature ranking determined with DALEX along the consecutive windows (axis X) processed over time. Window 0 covers the traces processed in initialization phase. Windows 1–28 cover the traces processed in the incremental phase.

As future work, we plan to explore how DALEX's explanations may be used for feature selection, to identify more stable features and improve the accuracy and robustness of the model. Additionally, we plan to investigate the effectiveness of intentional forgetting mechanisms, in order to maintain the freshness of the training set as older data begins to age. Finally, we intend to explore the use of online classification algorithms in the role of the oracle.

6 RELATED WORK

Network Intrusion Detection. In their seminal paper, Sommer and Paxson [56] reason about the intrinsic challenges of using machine learning to detect network attacks. After many years of successful ML-based approaches for detection of large-scale attacks such as worms [e.g., 61] and botnets [e.g., 29, 30], network attack detection research has struggled to make major breakthroughs in the past 10 years. INSOMNIA aims to reopen discussions on how machine learning can be adapted to overcome the specific challenges of the network intrusion domain (§2).

Learning with Drift. One of the major challenges of the network domain in the enterprise setting is extreme non-stationarity [8]. TESSERACT [48] quantifies severe concept drift in the malware domain, which causes rapid performance decay of ML-based detectors due to violations of the i.i.d. assumption. CADE [64] is a point-based outlier detector for drifting points, and has shown its effectiveness on three types of network attacks.



(a) Window 0 **(b) Window 1** **(c) Window 2**

(d) Window 22 **(e) Window 23** **(f) Window 24**

Figure 9: Top ten most-relevant features identified using the permutation-based variable-importance measure of DALEX [10] for windows 0, 1, 2 (top row), 22, 23, and 24 (bottom row). Arrows (↑↓) indicate features which changed in rank with respect to the previous window, while stars (⋆) indicate features which have newly entered the top ten. Dots (·) indicate no change.

The main objective of our work, INSOMNIA, is to mitigate concept drift in network scenarios through incremental learning and limited labeling cost. The temporal dimension is intrinsic in incremental learning and several works have been designed to exploit time-aware techniques. For instance, Mohamed et al. [43] propose a traffic anomaly detector relying on incremental learning that does not require model redeployment, but—unlike INSOMNIA—it requires partial ground truth labels for retraining over time.

Learning with Limited Labels. Given the difficulty in obtaining timely accurate labels for network traffic traces, semi-supervised and unsupervised learning methods are the most suitable solutions for deep learning-based intrusion detection systems. In the context of network intrusion detection, Taheri et al. [57] develop an unsupervised algorithm for outlier detection using incremental clustering. However, it focuses on point-based anomaly detection, whereas INSOMNIA aims to adapt to the distribution shift over time. Noorbehbahani et al. [45] propose a semi-supervised approach combining offline and online learning, while requiring some labels continuously over time. To limit the amount of labels required, active learning solutions are traditionally considered [51]. They rely on a subset of ground truth labels to retrain at each time step. In contrast to these previously mentioned approaches, INSOMNIA only requires labels at the initial training time, and then sustains itself without requiring manual labeling; instead, it generates pseudo-labels based on the nearest centroid neighbor.

Deep Learning for Intrusion Detection. A number of host-based intrusion detectors have been proposed using deep learning, e.g., DeepLog [22], Tiresias [53], and Shu et al. [55]. These defenses are complementary to network-based defenses and drift in their system-level event features is an orthogonal research problem.

The closest approach related to our work is Kitsune [42], a fully unsupervised network anomaly detector that leverages an ensemble of auto-encoders trained on benign data, with high reconstruction errors signaling attacks. However, we demonstrate that Kitsune's effectiveness diminishes drastically when faced with concept drift (Table 1). This is likely due to Kitsune being evaluated on IoT data which exhibited higher stationarity than typical enterprise network environments.

7 AVAILABILITY

We release INSOMNIA and the changes to TESSERACT [48] upon publication. We share it at https://bit.ly/insomnia_code for review.

8 CONCLUSION

We outline a set of open challenges facing modern ML-based intrusion detectors relating to a lack of uniformity in the distribution of traffic data over time. To tackle them, we propose INSOMNIA, a semi-supervised approach that uses active learning to reduce latency in the model updates, label estimation to reduce labeling overhead, and applies explainable AI to describe how the model evolves to fit the shifting distribution. We extend the TESSERACT framework [48] to perform a time-aware evaluation of INSOMNIA on a recently published, revised version of CICIDS2017 [23] and demonstrate that modern intrusion detection systems must address concept drift in order to be effective. We envision that future work may build on INSOMNIA in order to design robust intrusion detection models that can be sustained over time.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, ..., and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

[2] Charu C. Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and Philip S. Yu. 2014. Active Learning: A Survey. In *Data Classification: Algorithms and Applications*.

[3] Giuseppina Andresini, Annalisa Appice, and Donato Malerba. 2021. Autoencoder-based deep metric learning for network intrusion detection. *Information Sciences* (2021).

[4] Giuseppina Andresini, Annalisa Appice, and Donato Malerba. 2021. Nearest cluster-based intrusion detection through convolutional neural networks. *Knowledge-Based Systems* (2021).

[5] Giuseppina Andresini, Annalisa Appice, Nicola Di Mauro, Corrado Loglisci, and Donato Malerba. 2020. Multi-Channel Deep Feature Learning for Intrusion Detection. *IEEE Access* (2020).

[6] Giuseppina Andresini, Annalisa Appice, Luca De Rose, and Donato Malerba. 2021. GAN augmentation to deal with imbalance in imaging-based intrusion detection. *Future Generation Computer Systems* (2021).

[7] Annalisa Appice, Corrado Loglisci, and Donato Malerba. 2018. Active learning via collective inference in network regression problems. *Information Sciences* (2018).

[8] Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. 2017. Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing* 8, 2 (2017), 404–415.

[9] Stefan Axelsson. 2000. The Base-Rate Fallacy and the Difficulty of Intrusion Detection. *ACM Transactions on Information and System Security (TISSEC)* (2000).

[10] Hubert Baniecki, Wojciech Kretowicz, Piotr Piatyszek, Jakub Wisniewski, and Przemyslaw Biecek. 2020. dalex: Responsible Machine Learning with Interactive Explainability and Fairness in Python. *arXiv:2012.14406* (2020). https://github.com/ModelOriented/DALEX/

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[12] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proc. of the International Conference on Machine Learning (ICML)*.

[13] Avrim Blum and Tom M. Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proc. of the ACM Conference on Learning Theory (COLT)*.

[14] Leo Breiman. 2001. Random Forests. *Machine Learning* (2001).

[15] Anna L. Buczak and Erhan Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials* (2016).

[16] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-End Incremental Learning. In *Proc. of the European Conference on Computer Vision (ECCV) (Lecture Notes in Computer Science)*. Springer.

[17] B. B. Chaudhuri. 1996. A new definition of neighborhood of a point in multi-dimensional space. *Pattern Recognition Letters* (1996).

[18] François Chollet et al. 2015. Keras. https://keras.io.

[19] Bruno L. Dalmazo, João P. Vilela, and Marilia Curado. 2017. Performance Analysis of Network Traffic Predictors in the Cloud. *Journal of Network and Systems Management* (2017).

[20] Abebe Abeshu Diro and Naveen Chilamkurti. 2018. Distributed attack detection scheme using deep learning approach for Internet of Things. *Future Generation Computer Systems* (2018).

[21] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proc. of the International Conference on Information Systems Security and Privacy (ICISSP)*. SciTePress.

[22] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.

[23] Gints Engelen, Vera Rimmer, and Wouter Joosen. 2021. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*.

[24] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2019. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research (JMLR)* (2019).

[25] Prahlad Fogla, Monirul I. Sharif, Roberto Perdisci, Oleg M. Kolesnikov, and Wenke Lee. 2006. Polymorphic Blending Attacks. In *Proc. of the USENIX Security Symposium*.

[26] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR.

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks.. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. JMLR.

[28] Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised Learning by Entropy Minimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[29] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. 2008. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. (2008).

[30] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2007. Bothunter: Detecting malware infection through ids-driven dialog correlation.. In *USENIX Security Symposium*, Vol. 7. 1–16.

[31] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *Comput. Surveys* (2018).

[32] Roberto Jordaney, Kumar Sharad, Santanu Kumar Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Proc. of the USENIX Security Symposium*.

[33] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *Proc. of the International Conference on Learning Representations (ICLR)*.

[34] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *Proc. of the International Conference on Information Systems Security and Privacy (ICISSP)*. SciTePress.

[35] David D. Lewis and William A. Gale. 1994. A Sequential Algorithm for Training Text Classifiers. In *Proc. of the International ACM Conference of the Special Interest Group on Information Retrieval (SIGIR)*.

[36] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. 2012. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems With Applications* (2012).

[37] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2019).

[38] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. 2016. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks* (2016).

[39] Diego Marrón, Eduard Ayguadé, José R. Herrero, Jesse Read, and Albert Bifet. 2017. Low-latency multi-threaded ensemble learning for dynamic big data streams. In *Proc. of the IEEE International Conference on Big Data (Big Data)*.

[40] McAfee Labs. 2016. McAfee Labs Threats Report, December 2016. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2016.pdf.

[41] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, Anthony D. Joseph, and J. D. Tygar. 2016. Reviewer Integration and Performance Measurement for Malware Detection. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[42] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*.

[43] Marwa R. Mohamed, Abdurrahman A. Nasr, Ibrahim F. Tarrad, and Mohamed Z. Abdulmageed. 2019. Exploiting Incremental Classifiers for the Training of an Adaptive Intrusion Detection Model. *Int. Journal of Network Security* (2019).

[44] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaíz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognition* (2012).

[45] Fakhroddin Noorbehbahani, Ali Fanian, Sayyed Rasoul Mousavi, and Homa Hasannejad. 2017. An incremental intrusion detection system using a new semi-supervised stream classification method. *International Journal of Communication Systems* (2017).

[46] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2010).

[47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research (JMLR)* (2011).

[48] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proc. of the USENIX Security Symposium*.

[49] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. 2020. Tree-CNN: A hierarchical Deep Convolutional Neural Network for incremental learning. *Neural Networks* (2020).

[50] Burr Settles. 2012. Active Learning Literature Survey. *Synthesis Lectures on Artificial Intelligence and Machine Learning* (2012).

[51] Amin Shahraki, Mahmoud Abbasi, Amir Taherkordi, and Anca Delia Jurcut. 2021. Active Learning for Network Traffic Classification: A Technical Survey. arXiv:2106.06933 [cs.NI]

[52] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proc. of the International Conference on Information Systems Security and Privacy*

(ICISSP). SciTePress.
[53] Yun Shen, Enrico Mariconti, Pierre-Antoine Vervier, and Gianluca Stringhini. 2018. Tiresias: Predicting Security Events Through Deep Learning. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.
[54] Nathan Shone, Nguyen Ngoc Tran, Vu Dinh Phai, and Qi Shi. 2018. A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI)* (2018).
[55] Xiaokui Shu, Danfeng Yao, and Naren Ramakrishnan. 2015. Unearthing Stealthy Program Attacks Buried in Extremely Long Execution Paths. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*.
[56] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*.
[57] Sona Taheri, Adil M. Bagirov, Iqbal Gondal, and Simon Brown. 2020. Cyberattack triage using incremental clustering for intrusion detection systems. *International Journal of Information Security* (2020).
[58] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The Evolution of Android Malware and Android Analysis Techniques. *Comput. Surveys* (2017).
[59] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. 2018. A Survey on Deep Transfer Learning. In *Proc. of the International Conference on Artificial Neural Networks and Machine Learning (ICANN)*.
[60] R. Tibshirani, Trevor Hastie, B. Narasimhan, and Gilbert Chu. 2002. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proc. of the National Academy of Sciences (PNAS)* (2002).
[61] Ke Wang, Gabriela Cretu, and Salvatore J Stolfo. 2005. Anomalous payload-based worm detection and signature generation. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 227–246.
[62] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. 2020. Evaluating explanation methods for deep learning in security. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*.
[63] Ning Xie, Gabrielle Ras, Marcel van Gerven, and Derek Doran. 2020. Explainable deep learning: A field guide for the uninitiated. *arXiv preprint arXiv:2004.14545* (2020).
[64] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and Explaining Concept Drift Samples for Security Applications. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*.
[65] Yazhou Yang and Marco Loog. 2019. Single shot active learning using pseudo annotators. *Pattern Recognition* (2019).
[66] Chunlin Zhang, Ju Jiang, and Mohamed S. Kamel. 2005. Intrusion detection using hierarchical neural networks. *Pattern Recognition Letters* (2005).
[67] Xiaojin Jerry Zhu. 2005. *Semi-supervised learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

A IMPLEMENTATION

We develop INSOMNIA in Python 3, with the DNN architecture implemented using Keras 2.4 [18]—a high-level neural network API with TensorFlow [1] as the backend. The network comprises 3 fully-connected layers followed by two dropout layers to prevent overfitting. We apply the *ReLu* [27] activation to the hidden layers and to obtain output probabilities we use the softmax activation function on the final layer.

Weights are initialized following the Xavier scheme [26]—although during incremental learning, weights from the previous DNN model are used for fine-tuning each new model. The gradient-based optimization is performed using the Adam update rule [33].

Hyperparameters are optimized using the tree-structured Parzen estimator algorithm [11] as implemented in the Hyperopt library [12]. For optimization we perform a random stratified split to further divide the training set into training and validation sets at a ratio of 4:1, following the Pareto Principle. We select the hyperparameter configuration that achieves the lowest validation loss. The hyperparameter search space is reported in Table 2. For the No-Update baseline—also the vanilla DNN described in §1—we use this DNN alone without the update mechanism supplied by INSOMNIA.

For the Nearest-Centroid-based label estimation we use the classifier from the Scikit-learn [47] implementation. We measure the global feature relevance for the DNN using the DALEX [10] Python package 1.2.0.

Table 2: Hyperparameter search space for the DNN model.

Hyperparameter	Values
batch size	$\{2^5, 2^6, 2^7, 2^8, 2^9\}$
learning rate	[0.0001, 0.01]
dropout	[0,1]
#neurons per hidden layer	$\{2^5, 2^6, 2^7, 2^8, 2^9\}$

We extend the TESSERACT [48] framework to apply to the network intrusion detection setting. Notably we add functionality to temporally partition data using count-based windows rather than time splits, as well as a calculation of the impact of latency periods during model updates. Furthermore, we build on TESSERACT's post-classification stages for active learning to incorporate label estimation, fine-tuning, and explainability.

For the Kitsune [42] baseline we use the public implementation released by the authors. As the RMSE threshold we use the maximum RMSE of the benign traces during calibration. As the m parameter, i.e., the maximum number of inputs for any given autoencoder in the ensemble layer, we use the default size $m = 10$.