SoK: The Impact of Unlabelled Data in Cyberthreat Detection

Giovanni Apruzzese, Pavel Laskov, Aliya Tastemirova
Institute of Information Systems – University of Liechtenstein
{name.surname}@uni.li

Abstract—Machine learning (ML) has become an important paradigm for cyberthreat detection (CTD) in the recent years. A substantial research effort has been invested in the development of specialized algorithms for CTD tasks. From the operational perspective, however, the progress of ML-based CTD is hindered by the difficulty in obtaining the large sets of labelled data to train ML detectors. A potential solution to this problem are semisupervised learning (SsL) methods, which combine small labelled datasets with large amounts of unlabelled data.

This paper is aimed at systematization of existing work on SsL for CTD and, in particular, on understanding the utility of unlabelled data in such systems. To this end, we analyze the cost of labelling in various CTD tasks and develop a formal cost model for SsL in this context. Building on this foundation, we formalize a set of requirements for evaluation of SsL methods, which elucidates the contribution of unlabelled data. We review the state-of-the-art and observe that no previous work meets such requirements. To address this problem, we propose a framework for assessing the benefits of unlabelled data in SsL. We showcase an application of this framework by performing the first benchmark evaluation of SsL methods, which highlights the tradeoffs of 9 existing SsL methods on 9 public datasets. Our findings verify that, in some cases, SsL provides a small, but statistically significant, performance gain. This paper highlights that SsL in CTD has a lot of room for improvement, which should stimulate future research in this field.

Index Terms—machine learning, semisupervised learning, cybersecurity, labelling, threat detection

1. Introduction

Artificial intelligence and especially Machine Learning (ML) are one of the most important drivers of modern IT industry [1]. Specifically, in cybersecurity they can play a crucial role in several tasks, such as vulnerability analysis [2], security intelligence [3], and cyberthreat detection (CTD) [4].

As pointed out by Sommer and Paxson [5], the key to the tremendous success of the so-called supervised ML methods is their ability to build models connecting the training input data with the known ground truth information, often referred to as “labels”. It is well known, however, that obtaining the ground truth information in cybersecurity is a tough challenge [6]. In computer vision, even a child can say whether a picture shows a cat or a dog. In natural language processing, a linguist – albeit not always a layman – can easily assign linguistic tags to (parts of) the text or assess the quality of its translation. Precise characterization of security events is orders of magnitude more difficult and brings supervised ML methods to a dilemma: they work best when a model is built using an extensive dataset, but the price of labelling such a dataset may be outrageously high. According to Miller et al. [6], an entire company can only afford to label 80 malware samples per day.

On the other hand, unlabelled data is abundant in cybersecurity. Petabytes of network traffic at many levels, monitoring and endpoints logs, as well as many other information sources (e.g., threat intelligence feeds [7]) can be utilized for CTD tasks. Unlabelled data can be used in unsupervised ML methods. For instance, clustering proved to be successful for some cybersecurity applications, e.g., analysis of malware families [8]. However, such techniques can only address ancillary tasks, and cannot automate any detection (i.e., classification) mechanism without large amounts of labels that clearly distinguish benign from malicious samples (e.g., [9]).

Semisupervised learning (SsL) [10] has a promise to deliver efficient ML classifiers that require small amounts of labelled data by exploiting the information gained from large sets of unlabelled data. For instance, in self learning (e.g., [11]), labelled data is mixed with unlabelled data to improve performance; similarly, in active learning [12] an initial classifier trained on a small labelled dataset can be used to analyze a large set of raw data, and then ‘suggest’ the most cost-effective samples to label. The SsL setup resembles typical cybersecurity scenarios and, as a result, many works (e.g., [13]–[15]) proposed SsL solutions for diverse CTD tasks.

In a recent study from computer vision, Oliver et al. [16] state, however, that “the gap in performance between SsL and using only labelled data is smaller than reported”, and that “a classifier trained on a small labeled dataset with no unlabeled data can reach very good accuracy”. These observations raise the question: is unlabelled data indeed beneficial to SsL and, if so, is it worth the effort? Unlabelled data is cheaper than labelling, but it is not completely ‘free’: e.g., its processing and storing costs are not negligible [17], [18].

We researched previous works utilizing the combination of labelled and unlabelled data for CTD and found—surprisingly—that none of such works addressed the benefits of unlabelled data in SsL. Identifying such benefits is not simple, because it requires the analysis of the key components that contribute to the development of SsL solutions. Therefore, to the best of our knowledge, it is still unclear whether SsL is advantageous for CTD. This is because prior works adopt evaluation protocols that are not standardized and do not allow to assess whether unlabelled data is cost-effective. Such immaturity serves as the main motivation for our SoK, and our specific goal is to promote
deployment of SsL methods.

To this end, we formalize a set of requirements derived by a systematic analysis of a realistic deployment of SsL methods in CTD. Following our requirements, it is possible to assess the impact of unlabelled data on the quality of SsL models. Moreover, we propose a novel evaluation framework that can be used to assess existing and future SsL methods in a research environment. Finally, we showcase an application of our framework to perform the first ‘benchmark’ where we statistically verify the benefits of well-known SsL methods on 9 publicly available datasets for diverse CTD tasks.

Since the scope of this work is well beyond the traditional systematization of previous work, let us outline the structure and the contributions of this paper.

We begin with the presentation of the background and related research fields in §2. The main focus of this presentation is to illustrate the challenges of labelling data in CTD, which is significantly more difficult than in other domains and, hence, motivates the search for solutions that can work with only a small amount of ground truth.

In §3, we formally present the general goal of SsL and analyze its cost structure. The main contributions of this section are an original cost model for SsL in CTD, the definition of the benefit of unlabelled data in SsL, and the set of requirements that must be met by research evaluations to ensure that such benefit can be claimed.

In §4, we review the state-of-the-art on SsL for CTD, and analyze to what extent each paper meets the requirements identified in §3. The main conclusion of this analysis is that no prior work can satisfy all the requirements, and hence demonstrate the benefits of using unlabelled data. This finding must not be interpreted as a deficiency of prior work; it merely highlights that no one ever questioned the utility of unlabelled data in the deployment of SsL for CTD.

As a first step towards such deployment, we present a new evaluation framework for SsL in §5 and demonstrate its application by assessing 9 well-known SsL methods using 9 datasets in §6. All these contributions provide a constructive approach for assessing the utility of unlabelled data and promote the rollout of SsL for CTD.

Finally, we discuss our findings (§7) and outline the conclusions of our study (§8).

2. Background and Related Work

Obtaining ground truth labels for training ML models is a well-known problem which motivates the investigation of SsL methods (e.g., [19], [20]). However, with respect to other application domains of ML, the process of labelling in CTD is fundamentally harder, making SsL methods particularly attractive here. We elucidate all these labelling difficulties, representing the main motivation of our paper.

2.1. Uniqueness of CTD with Respect to Labelling

In some application domains of ML, there exist natural factors that facilitate acquiring labelled data for training ML models. Such factors can be data sharing (e.g., [21]), inherent low cost of labelling (e.g., the popular CAPTCHAs [22]), or long-term usage of labelled data (e.g., ImageNet was collected in 2009 and is still widely used today [23]).

All of these factors are not applicable to cybersecurity due to two intrinsic characteristics. First, the intrinsic confidentiality which strongly discourages data sharing. Second, the constant adaptation of attackers as well as the growth and the evolution of the environments being protected lead to the phenomenon known as concept drift [24], i.e., a fundamental change (gradual or abrupt) of the respective data generation processes. This latter issue is crucial, as it conflicts with the underlying ‘iid’ assumption of ML and hence adversely affects its reliability in production environments [5].

The peculiarities of CTD are especially clear in comparison to computer vision. In image recognition problems, the underlying ground truth is clear and stable. “A cat will always be a cat, whereas a dog will always be a dog” [25], and a person can usually distinguish between two image classes very well [26], although specific applications may demand more informed opinions (e.g., physicians for cancer diagnoses [27]). Moreover, after acquiring such labelled data, it is possible to apply data augmentation [28] strategies to increase its effectiveness. For instance, adding some noisy pixels or mirroring the image allows one to create a new image that is different from the original but still has the same ground truth.

In contrast, all of the following can occur in CTD:

- a malicious sample is benign elsewhere [5];
- a malicious sample can be purposely crafted to represent a benign sample [29];
- a benign sample today becomes a malicious sample tomorrow (the so-called ‘label shift’ [30]).

To aggravate the problem, verifying the ground truth of a sample is hard even for security experts, requiring further verifications [31]. Finally, data augmentation is difficult to apply in CTD: for instance, changing a single byte can turn many malicious samples into benign samples [32].

Without loss of generality, we can state that a dataset that is usable for realistic CTD applications of ML must meet the following criteria [33]–[35]:

- It must be large enough to capture all the underlying characteristics of the environment to protect, and of the threats to defend against [5].
- The ratio of benign/malicious samples must be balanced enough to allow efficient detection without generating excessive false alarms [36].
- It must have accurate ground truth [31].
- It must be continuously updated [24].

These peculiarities make obtaining labelled datasets for CTD a tougher challenge than in other domains.

2.2. Specific Labeling Issues in CTD

We analyze the common procedures (and respective issues) for labelling data in CTD, split in three broad areas [4], [37]: Network Intrusion Detection (NID), Phishing Website Detection (PWD), Malware Detection (MD).

Network Intrusion Detection. The detection of intrusions within a network perimeter can greatly benefit from ML [4], [38] when labelled datasets are available.

1. \textit{iid}=independent and identically distributed random variables.
In the case of NID, such datasets contain samples providing network-related data. Well-known formats include full Packet Captures (PCAP) or Network Flows (NetFlows) [39]. PCAP provides low-level information but it is not usable if the traffic is encrypted\(^2\); moreover, storing and analyzing PCAP is computationally expensive. In contrast, the NetFlow format mitigates these issues by providing a high-level overview of network communications between two endpoints while still enabling an appreciable detection performance [39], [40]. Other common data formats are DNS records for investigation of malicious domains [41], and SNMP for monitoring of specific hosts [42]. Regardless of the data-type, a common problem in NID is that every network is unique [5], [36]. An anomalous behavior in one network may be normal in another network, hence preventing a reliable ‘transfer’ of ML models. Moreover, obtaining accurate ground truth is tough for both benign and malicious samples. Most existing NID datasets used in research are created by infecting some machines in a controlled network environment with known malware, and capturing the traffic of the simulated network (e.g., [43]–[45]). Such approach is difficult to apply in reality. Verifying that a sample is truly legitimate requires ensuring that both hosts (the source and destination) ‘connected’ by the specific traffic sample are not malicious. If one of these hosts is compromised via an unknown vulnerability then labelling all the traffic generated by such host as benign can lead to poisoning attacks [29]. Acquiring malicious samples is also difficult, because it requires compromising real machines with malware, hence exposing the network to external threats. Another problem is to consider all traffic originating from an infected machine is malicious, as some of its data may be generated by legitimate network activities (e.g., ARP messages). All such challenges increase the difficulty of obtaining representative datasets for NID [46].

**Phishing Website Detection.** Phishing attacks can be launched in various ways, e.g., via email or social networks [47]. In this paper, we focus on the detection of phishing websites because phishing usually involves luring the victim to enter some information on a (phishing) website. Attackers can easily create ‘squatting’ websites that are difficult to detect [48], making them a rampant threat [49]. Datasets for PWD may include diverse data derived from, e.g., the URL (its length or the usage of some characters), the DNS record (a recent website is more likely to be a phishing hook), the HTML code (phishing websites have many pointers to external domains), and even the image of the landing page (most phishing hooks are similar to legitimate websites) [50]–[53], each having its pros and cons [54]. The most common way to create labelled datasets for PWD is to use public lists of benign or malicious pages (e.g., AlexaTop or PhishTank). The webpage can be visited and the relevant information extracted to compose a given dataset [55]. In general, this process makes labelling for PWD easier than in NID, because verification is simple through expert knowledge [56]. Moreover, the ground truth of a website is independent on the target system (phishing page will ‘always’ be malicious), enabling model transfer. However, such advantage presents some risks. For instance, the PWD embedded in the Google’s Chrome web-browser was reverse engineered, allowing to craft phishing webpages that bypass detection [57]. Therefore, transferring ML models without any influx of new labelled data can result in unreliable PWD.

**Malware Detection.** The advent of data-driven solutions, such as ML, allowed the identification of malware variants which bypass traditional rule-based systems [9]. Even commercial products leverage ML [38], which can be used for both static and dynamic MD [59], [60]. The complexity of labelling data for MD falls in between PWD and NID. Obtaining a large corpus of files (benign or malicious) for MD is not difficult per-se, as it can be done via public repositories. However, relying on such data without verifying the ground truth is risky. For instance, well-known marketplaces were recently found to contain malicious applications [61]. Treating such applications as benign exposes to poisoning attacks, and hence further verifications are required. Such verifications are, however, challenging: some web-services can automatically analyze an input (e.g., VirusTotal), but such services can disagree [37], leading to unreliable results that require costly validation by experts [52]. A possible workaround is using ‘collective wisdom’ techniques that consider diverse antimalware engines [63]; recent works also propose to sanitize ‘noisy’ labels in the training data [64].

| Takeaway | Acquiring labelled data for CTD is challenging. To facilitate the development of ML-solutions, it is necessary to investigate approaches that can work when most of the data is not provided with the ground truth, such as SSL methods. |

| 2.3. Focus of the Paper |

Combining labelled and unlabelled data to improve the proficiency of ML methods can be done in many ways. This SoK paper focuses on development of ML-systems for CTD, devised by using (i) small sets of labelled data (ii) together with large sets of unlabelled data. Such settings align with the definition of ‘Semisupervised Learning’ by Oliver et al. [16]. To clarify our focus, we describe two exemplary techniques, illustrated in Fig. 1: self learning via pseudo-labelling (e.g., [65]) and active learning via uncertainty-sampling (e.g., [66]), both associated with SSL [67]–[69].

![Figure 1: Active Learning and Pseudo Labelling.](image)

In self learning, the task of a ML model is to automatically learn from itself. In the specific case of pseudo labelling, the intuition is to first train a ML model on a
(small) set of labelled data (L in Fig. 1), and then use such model to predict the label of a (large) set of unlabelled data (U in Fig. 1). Such ‘pseudo-labels’ are then used to retrain the model on a mixed dataset, containing both the correct labels of L and the pseudo-labels of U. To avoid using potentially wrong labelled data, the retraining can be done by using only the pseudo-labelled samples with a high confidence estimated by the model.

In active learning, the ML model interacts with an oracle (realistically, with a human) to improve its learning phase. In the specific case of uncertainty sampling, the idea is to first train a ML model on a (small) set of labelled data, L, and then use such model to analyze a (large) set of unlabelled data, U. The analysis is focused on ‘suggesting’ to the oracle which samples in U should be correctly labelled to improve the performance, and the suggestion is based on the confidence of the model on the samples in U. Intuitively, the model can learn ‘more’ from samples with a low confidence [70]. The oracle then assigns the correct ground truth to such samples, which are inserted into L and used to retrain the model—using a correctly labelled dataset.

2.4. Related Work

Many learning paradigms focus on providing reliable models under the assumptions of scarce label availability; often, the term ‘Semisupervised Learning’ is used to describe techniques that deviate from our definition. We summarize a few orthogonal areas to our work.

Federated learning [71] aims to develop a ‘global’ model by combining ‘local’ models trained on small datasets. Despite some risks (e.g., poisoning [72]) recent efforts applied it successfully even in privacy-sensitive settings [73]. However, federated learning makes no use of unlabelled data, and the final labelled dataset is huge.

Few-shot learning has the goal of identifying unknown classes when only very few (or even zero [74]) labels are available. For example, by finding the most relevant parameters of a baseline feature extractor, it is possible to generalize on unseen classes [75]. These approaches—conceptually similar to ‘anomaly detection’—are thus tailored for detecting novel attacks, and showed promising achievements even in CTD (e.g., [76]). However, they assume a large amount of initial labelled samples for the ‘known’ classes. As an example, the authors of [77] train their one-class classifiers on 80% of the available samples for the ‘normal’ class. Similarly, the ML-NIDS in [74] can detect 14 ‘unseen’ attacks, but is trained on over 200k samples spanning over 12 ‘known’ classes.

Representation learning focuses on finding the features that maximize the performance of a ML classifier [78]. It is possible to use unlabelled data to fine-tune such selection (e.g., [79]–[83]). Such procedures are ancillary to detection tasks, and hence outside our scope. As an example, the anomaly detector in [78] mixes a large set of unlabelled data with a small set of labels to identify the most representative features: however, once such features are identified, the experiments for the actual ‘detection’ are performed by using 60% of the (fully labelled) available data to train the classifier.

Finally, lifelong learning aims to update ML models over-time [84], [85], which can be done by exploiting future (unlabelled) data streams; an assumption shared by active learning. However, sometimes the initial cost of labelling is neglected (this is the same problem as few-shot learning). An exemplary case is Tesseract [86], focusing on time-aware MD. The idea of Tesseract is using, among others, active learning strategies to improve over time: despite increasing performance by 20% with 700 additional labels, the initial deployment of Tesseract requires huge amounts of correct labels (i.e., more than 50K). Hence, some applications of active learning may have assumptions that deviate from ours.

We stress that our paper focuses on CTD. As such, any proposal that uses SsL for a different security-related task (e.g., fingerprinting [87]) is orthogonal to this paper.

3. Semisupervised Learning for CTD

In contrast to obtaining verified labels, acquiring unlabelled data for CTD is relatively straightforward.

Semisupervised Learning (SsL) aims to combine unlabelled with labelled data to devise ML models, which leads to the following question: “what is the benefit of unlabelled data?”. Only by answering this question it is possible to understand the role that SsL can play in CTD.

Let us begin our research by introducing SsL and its relationship with traditional supervised learning (SL).

3.1. Introduction to Semisupervised Learning

Any ML model requires to learn from data so that, after its deployment, it can efficiently analyze future and unseen data, which we denote as F. For those ML methods that require supervision (such as SL and SsL), the learning is done by means of labelled data. To achieve the best performance, supervised models should be trained on a huge and fully labelled dataset, L. Creating such L may, however, be prohibitive. In contrast, under the constraint of a limited labelling budget, C, the amount of labelled data will be smaller, L, and the resulting model may be inferior to the model that could have been built using L.

SsL aims at bridging the gap between labelling effort and the model quality, by using L alongside a large unlabelled dataset, U, which is cheap to acquire. The resulting model should attain a better performance on F than a model built with same L but without using U.

For instance, consider the two methods described in §2.3. In self-learning, the model should achieve a better performance after retraining on the ‘pseudo-labels’ from U than the initial learner. In active learning, the model trained on the L with the ‘suggested’ samples from U should outperform a model trained on a L derived from the same labelling budget C, but without any ‘suggestions’ derived by U.

We now formally define the abovementioned scenarios. Without loss of generality, any CTD task can be seen as 1+N classification ML problem, where samples are either benign, or belong to one among N malicious classes. For simplicity, in the remainder we will consider a binary-classification setting; all our considerations can be extended to cover also multi-classification settings.

Let L be a given labelling budget. Let L be any labelled dataset containing sample-label pairs, obtained
by using $L$. The composition of any labelled dataset is determined by its size, $|\cdot|$, and its class balance ratio, $\rho(\cdot)$, which is a 1+N dimensional vector defining the distribution of its samples in percentages. Let $|L|$ and $\rho(L)$ denote the size and class balance ratio of $L$. Let $\mathbb{L}$, with $\rho(\mathbb{L})$, be a superset of $L$.

Let $\mathbb{U}$ be an unlabelled dataset containing samples of which the ground truth is not known, with $|\mathbb{L}| \ll |\mathbb{U}|$.

Finally, let $\mathbb{F}$ be another labelled dataset whose $|\mathbb{F}|$ and $\rho(\mathbb{F})$ should enable meaningful performance assessments. Because $\mathbb{F}$ represents future data, $\mathbb{F} \cap (\mathbb{L} \cup \mathbb{U}) = \emptyset$. Let $\mu$ be any performance metric, e.g., accuracy or F1-score.

An illustration of the upcoming definitions is in Fig. 2.

![Figure 2: Semisupervised Learning w.r.t. Supervised Learning.](image)

A Supervised Learning (SL) method uses a labelled dataset to train a model that, after deployment, can predict the ground truth of unseen samples (i.e., in $\mathbb{F}$) obtaining a certain performance $\mu(\cdot)$. To achieve optimal performance, the dataset used to train a model via SL should be comprehensive, that is, it should be adequately large and balanced. As a direct consequence, a model trained on a very small dataset will have a subpar performance. Let $\mathbb{SL}$ be a model trained on $\mathbb{L}$, and $\mathbb{SL}$ be a model trained on $\mathbb{L}$: if $|\mathbb{L}| \ll |\mathbb{F}|$ then $\mu(\mathbb{SL}) < \mu(\mathbb{SL})$, irrespective of $\rho(\mathbb{L})$. In the remainder, we assume that training $\mathbb{SL}$ on $\mathbb{L}$ results in optimal $\mu(\mathbb{SL})$, and that $|\mathbb{L}| \ll |\mathbb{F}|$.

We can now provide the following definition:

**Definition 1.** The goal of a Semisupervised Learning (SsL) method is using $\mathbb{U}$ alongside any $L$ obtained with $\mathbb{L}$ to devise a model $\mathbb{SsL}$. After deployment, $\mathbb{SsL}$ should predict the ground truth of the samples in $\mathbb{F}$ by achieving a performance $\mu(\mathbb{SsL})$ s.t.: $\mu(\mathbb{SL}) < \mu(\mathbb{SsL}) \leq \mu(\mathbb{SL})$.

From Def. 1, we derive that assessing the benefits of SsL is linked with SL, because only by comparing SsL with SL (and $\mathbb{SL}$) the impact of $\mathbb{U}$ can be determined.

### 3.2. Cost model of SsL for CTD

Let us interpret the abstract descriptions in §3.1 from the perspective of a real organization willing to deploy a SsL solution for CTD. We provide an illustration in Fig. 3.

![Figure 3: Proposed cost model for deployment of SsL methods.](image)

In the real world, a resource investment is required for integrating any new solution, and the decision to deploy such a solution depends on its estimated return on investment (ROI). Assessing the ROI demands the definition of a cost model that allows practical comparisons to support operational decision making.

Huge amounts of data, $\mathbb{U}$, can be easily obtainable. An organization can have a rough idea about the nature of such data, but the impossibility of determining the ground truth without costly manual inspection makes it necessary to treat all its samples as unlabelled.

However, any SsL model necessitates at least a small amount of labelled data, $\mathbb{L}$. Such $\mathbb{L}$ can be acquired either before and independently from $\mathbb{U}$, or by using $\mathbb{U}$. For instance, $\mathbb{L}$ can be obtained by verifying uncertain labels, manually labelling existing data, or even by creating new labelled data via controlled simulations (§2). In addition, like all ML-solutions, SsL models require ancillary operations (e.g., feature engineering, training and tuning).

We can hence define the cost model for developing SsL solutions. Deployment of a semisupervised model $\mathbb{SsL}$ requires to invest some budget $\mathbb{B}$. Such budget can be seen as the sum of three elements $\mathbb{U}$, $\mathbb{L}$, and $\varepsilon$. Specifically:

- $\mathbb{U}$ represents the investment for obtaining (and, if necessary, maintaining) the unlabelled data $\mathbb{U}$;
- $\mathbb{L}$ is the investment for generating $\mathbb{L}$. Specifically, $\mathbb{L}$ is used to ensure that any sample $x \in \mathbb{L}$ has the correct label. Such $\mathbb{L}$ can be used at all once or at different times: for instance, it is possible to reserve a portion for subsequent labelling rounds that depend on $\mathbb{U}$ (e.g., active learning);
- $\varepsilon(\mathbb{SsL})$ is used for any extra operation for developing $\mathbb{SsL}$ that is not related to labelling the samples in $\mathbb{L}$. For instance, $\varepsilon$ can be used to (i) conduct preliminary analyses on $\mathbb{U}$, (ii) tune $\mathbb{SsL}$, (iii) process data, as well as for any (iv) computational costs.

By using $\mathbb{L}$, the organization eventually obtains a labelled dataset $\mathbb{L}$, whose composition (i.e., $\rho(\mathbb{L})$ and $|\mathbb{L}|$) depends on the cost of labelling each individual sample $x$ in $\mathbb{L}$, let $C_x$ denote such cost. We can express our cost model by formally defining $\mathbb{B}(\mathbb{SsL})$ with the following Equation:

$$\mathbb{B}(\mathbb{SsL}) = \mathbb{U} + \mathbb{L} + \varepsilon(\mathbb{SsL}), \text{ where } \mathbb{L} = \sum_{x \in \mathbb{L}} C_x \text{ (Eq. 1)}$$

6. ToBooK, we are the first to propose a SsL-specific cost model.
3.3. Requirements for Evaluation of SsL Methods

Let us explain how to conduct evaluations that allow to estimate the ROI and, hence, gauge the benefits of SsL.

In research, evaluations of ML are done by means of fully labelled datasets, which should represent realistic scenarios and hence must be unbiased. Let \( \mathcal{D} \) be one of such datasets\(^7\). From Def. 1, a SsL model aims at \( \mu(SL) < \mu(SsL) \leq \mu(SL) \). Therefore, the source dataset \( \mathcal{D} \) must be used to derive the four sets \( \mathcal{F}, \mathcal{U}, \mathcal{L}, \mathcal{L} \) necessary to compute such performance. To align the evaluation with real deployment use-cases, the creation of these sets must be done by taking \( \mathcal{B} \) and, hence, \( \mathcal{L} \) into account. Note that \( \mathcal{F} \) is only used to assess the performance on simulated ‘unknown’ data, which is not available in advance. Therefore, using \( \mathcal{F} \) to ‘cherry pick’ the samples to put in \( \mathcal{L} \) is prohibited\(^{[90]} \).

We can now define the 7 requirements, applicable to any CTD task, that must be upheld to ensure unambiguous assessment of the benefits of SsL methods.

**Req. 1 (Lower Bound).** It is necessary to evaluate a lower bound model that only uses \( \mathcal{L} \) and makes no use of \( \mathcal{U} \). In other words, train a model \( SL \) on \( \mathcal{L} \) and evaluate its performance on \( \mathcal{F} \) as \( \mu(SL) \). To avoid bias\(^8\), such \( \mathcal{L} \) must be chosen by random sampling from \( \mathcal{D} \) subject to \( \mathcal{L} \). **Motivation:** The \( SL \) represents the true baseline\(^9\), allowing to assess the benefit (and, hence, compare the ROI) of any model \( SsL \) that uses \( \mathcal{L} + \mathcal{U} \). For instance, if \( \mu(SsL) \approx \mu(SL) \) there is no practical benefit in using the unlabelled data in \( \mathcal{U} \); it may also be that \( \mu(SsL) < \mu(SL) \), meaning that using \( \mathcal{U} \) is detrimental.

**Req. 2 (Ablation Study).** It is necessary to always consider a ‘vanilla’ model \( SsL \) that uses \( \mathcal{U} \) in a trivial way together with an \( \mathcal{L} \) randomly sampled from \( \mathcal{D} \). The aim is minimizing the degree of supervision\(^{10} \) involved by using \( \mathcal{U} \). **Motivation:** The ‘vanilla’ \( SsL \) allows to gauge (i) the smallest improvement provided by \( \mathcal{U} \) via comparisons with \( SL \); and also (ii) the smallest cost induced by using \( \mathcal{U} \), because the randomness of \( \mathcal{L} \) and the lack of supervision makes the corresponding \( \varepsilon \) minimal. Moreover, \( SsL \) serves as a baseline for an ablation study\(^{[20]} \), to simulate worst case scenarios in which any operation that relies on \( \mathcal{U} \) to refineably compose \( \mathcal{L} \) is not functional in practice.

**Req. 3 (Upper Bound).** It is necessary to train an upper bound model \( SL \) on \( \mathcal{L} \) and evaluate its performance on \( \mathcal{F} \) as \( \mu(SL) \). **Motivation:** The \( SL \) serves to assess the performance achievable by augmenting \( \mathcal{L} \) until it reaches \( \mathcal{L} \). Moreover, if \( \mu(SL) \approx \mu(SL) \), then investing in \( SsL \) may not be worth it in the first place.

**Req. 4 (Statistical Significance).** It is necessary to verify the statistical significance of any evaluation result. **Motivation:** Evaluations of SsL involve a lot of randomness and uncertainties. The huge search space for composing \( \mathcal{L} \) from \( \mathcal{D} \) may lead to erratic results: in some cases \( \mu(SsL) > \mu(SL) \), but the opposite can also be true. Moreover, sometimes \( \mu(SsL) \approx \mu(SL) \) meaning that further tests are required to determine if \( SsL \) is superior to \( SL \). Hence, different draws of \( \mathcal{L} \) and, preferably, also of \( \mathcal{L} \) and \( \mathcal{F} \) must be assessed and conclusions must be drawn after statistically significant comparisons\(^{11} \).

**Req. 5 (Transparency).** It is necessary to ensure full transparency on the composition of \( \mathcal{L}, \mathcal{L}, \mathcal{U}, \mathcal{F} \). This implies: specifying size of each dataset (both in absolute numbers, and with respect to \( \mathcal{D} \)); and the balance ratios in terms of class composition. **Motivation:** Claiming that a given SsL method is effective when using only “1% of the data” is not as enticing if \( \mathcal{D} \) has 1M samples. Moreover, the balance ratio can significantly alter the results when small datasets are considered. This requirement also serves to estimate \( \mathcal{L} \) and \( \mathcal{U} \).

**Req. 6 (Reproducibility).** Any evaluation must be supported with information that allow its reproducibility\(^{[91]}, [92] \). **Motivation:** aside from obvious reasons, it serves to approximate \( \varepsilon \).

**Req. 7 (Multiple Settings).** It is necessary to evaluate any model by considering multiple deployment settings, e.g., diverse datasets. **Motivation:** for practical deployments, the samples in \( \mathcal{D} \) must resemble the true distribution at inference, and CTD scenarios can vary (cf. \( \S 2 \)).

---

7. \( \mathcal{D} \) can be subject to some preliminary preprocessing.
8. Labelling is not deterministic, and in realistic scenarios it is not possible to know in advance the effectiveness of labelling.
9. Such \( SL \) should aim at maximizing the gain from \( \mathcal{L} \) (and hence \( \mathcal{L} \)), but it should not be overtuned (leading to overfitting and astronomical \( \varepsilon \)), nor it should be ‘misconfigured’ to inflate results.
10. Depending on the context, there can be many ways of devising \( SsL \). As an example, in pseudo-labelling, a ‘trivial’ way is using all pseudo-labels regardless of their confidence; whereas, in active learning, a ‘trivial’ way is labelling randomly chosen samples (instead of those with least confidence).
11. Such efforts go beyond traditional ‘cross-validations’\(^{[86]} \).
The intuition behind our requirements is that any model SsL developed with a given SsL method must be compared against the three baselines of Regs. 1–3. By doing so, it is possible to measure the added value of SsL, building on U, potentially creating a better L, but with a specific extra cost ε. It is implicit that any model-to-model comparison must be done under the assumptions of identical L.

We can now formally define the benefit of U in SsL.

**Definition 2.** Unlabelled data U used to develop any SsL is **beneficial** if it is shown that: (i) \( \mu(SsL) \ll \mu(SL) \), and (ii) ROI(SsL) is better than both ROI(SL) and ROI(U).

Since SL does not use U, and SsL should minimize ε (w.r.t. SsL), it follows that \( \mu(SsL) \) must be greater than \( \mu(SL) \) and also greater or equal than \( \mu(U) \). To justify deployment of a new SsL method, its evaluation must show that Def. 2 holds in different settings.

**Takeaway:** Evaluating SsL in research requires to train and test (i) multiple models (ii) many times and in (iii) different scenarios, while reporting all details of the experimental setup. By meeting these requirements it is possible to assess the deployment benefits of a SsL method.

### 4. State-of-the-Art

We analyze the state-of-the-art w.r.t. the proposed requirements (§3.3), and provide a summary in Table 1. Let us describe our methodology, and then discuss the main findings.

#### 4.1. Methodology

To assess the extent to which existing works meet our requirements, we perform a systematic literature survey. Such process is organized in three phases: **search**, **screening**, **investigation**. To reduce bias, all phases involved two researchers who worked independently, and who discussed their individual findings in weekly meetings.

**4.1.1. Search.** We first searched for all literature linking (even remotely) SsL with CTD in well-known scientific repositories. Such repositories include IEEE Xplore, Google Scholar, and ACM Digital Library; but we also considered the proceedings of top security conferences. In particular, we searched for the following keywords:

\[
\text{(semi-supervised} \lor \text{semisupervised} \lor \text{semi supervised} \lor \text{active}) \land \\
\text{(network} \lor \text{malware} \lor \text{phishing} \lor \text{intrusion})
\]

which had to be included either in the title or in the abstract. Any work that was not peer-reviewed was excluded, and we looked for papers published after 2007. The results of such search formed an initial corpus of papers, which was further extended with all papers that either cited, or were cited by, a given work (and that included same or similar keywords); as well as with papers that the authors autonomously found during their daily duties (e.g., reviewing).

**4.1.2. Screening.** After obtaining the corpus of candidate papers, we studied them with the intent of determining which papers fall within our scope. By referring to §2.3, a paper had to meet three criteria: (i) focus on CTD, (ii) using unlabeled data, (iii) in combination with small sets of labelled data\(^2\). After several discussions between the two researchers, this phase resulted in the set of 48 papers reported in Table 1. To the best of our knowledge, Table 1 represents the current state-of-the-art of SsL for CTD.

**4.1.3. Investigation.** Those papers that met all the inclusion criteria were then further analyzed, with the goal of assessing their compliance with the proposed set of requirements. The results are in Table 1, which is organized as follows. We distinguish the papers on the basis of the three main CTD areas of interest (NID, PWD, MD); cells with a gray background denote papers that specifically consider ‘active learning’ approaches. For each paper we compare it with our requirements: a ✓ (resp. ✗) denotes that a requirement is met (or not).

- For Req. 1, we use ✓ if it is not explicitly mentioned that \( L \) was randomly drawn, and ✗ if either the SsL is missing, or when such SsL uses a different \( L \).
- For Req. 2, we use ✓ if the paper considers a SsL model that is completely unbiased and can serve as an ablation study; ✗ if the SsL models are trained on an unbiased random \( L \), but cannot serve as an ablation study due to ‘overuse’ of U; and ✗ if the provided information is insufficient to determine the absence of bias in \( L \).
- For Req. 3, is binary, but we also consider as ✓ when SsL is trained on a very large set of correct labels.
- For Req. 4, use ✓ if only some ‘cross-validation’ is performed, ✗ if statistical comparisons are made or mentioned, and ✗ otherwise.
- For Req. 5, we report two columns: ‘Labels’ denotes whether the provided information allows to determine the actual number of labelled samples used to train and test all the considered models; ‘Balance’ denotes whether the balancing ratios are clearly specified.
- For Req. 6, ✗ denotes if the provided information is sufficient for reproduction; ✓ if the source code is open; and ✗ if only intermediate information is provided.

In the last column we report the datasets used in each paper: here, ‘Private’ means that the data was never made available, whereas ‘Custom’ means that it was composed in-house via public sources, but that the actual samples cannot be recovered (i.e., it is not possible to retrieve the public feeds of past years).

#### 4.2. Findings

By observing Table 1, we derive the following.

1. No one fits all: no paper meets all requirements.
2. Few compare their SsL methods with a lower bound.
3. Worst case scenarios are rarely covered.
4. Lack of statistically significant comparisons, preventing any certification of the final results.
5. Poor reproducibility and limited datasets, which is a known trend in ML research [128].

\(^{12}\) The term ‘Semisupervised Learning’ has many meanings (§2.4).
Although no paper meets all our requirements, there are some good efforts. Remarkably, Zhao et al. [111] meet most almost all requirements (with the exception of a single dataset, and their implementation not being available today), and are the only ones to mention a statistical comparison via a student t-test; however, we were not able to infer how the starting dataset was split in training and testing. We also praise the work in [111], but it lacks a vanilla SsL for ablation studies (due to fine tuning of confidence thresholds), and only a limited cross-validation is performed. Noteworthy is also [109], whose authors fairly evaluate different SsL techniques (although none of these can be considered as an ablation study) for two CTD tasks (NID and MD), but each on a single dataset. The situation portrayed by Table 1 does not imply that all past works are wrong or flawed; these works are published in high quality venues, and we acknowledge their significance. On the contrary, the true message of Table 1 is highlighting the immaturity of the state-of-the-art with respect to realistic deployments of SsL. No attention has been given to systematic assessments of the benefits provided by unlabelled data in SsL.

The case of active learning. We observe that many papers in Table 1 use active learning methods, most of which in lifelong learning settings. In these cases, considering a model trained via random draws from $\mathbb{U}$ (instead of ‘active’ suggestions) can simultaneously meet both Req. 1 and 2. This is notably done by Gornitz et al. [95] (despite not meeting Req. 3). In contrast, Pendlebury et al. [86] apply active learning by labelling all samples with confidence below 1%, which results in 700 samples, and the improvement is shown against a model that does not make use of any additional label, leading to an unfair comparison. The evaluation protocol of these and similar papers is, however, legitimate. They operate under the assumption that the ML model is already trained and deployed, meaning that unlabelled data will naturally occur. In such conditions, the focus is not on “determining the benefits of unlabelled data for ML deployment” (which is our focus), but rather on “how to maximize the performance of an existing ML system with additional unlabelled data streams”. Both [86] and [95] achieve that. Nonetheless, to the best of our knowledge (semi)supervised ML systems are not widely deployed (yet) in CTD, demanding further investigations of their potential benefits in advance.

Relationship with other domains. There are several studies that expose evaluation issues of ML methods, and Delgani et al. [129] invite devising specific guidelines. However, within the context of SsL, existing proposals are not applicable to CTD. For instance, Oliver et al. [16] suggest transferring models between different datasets: this may not be feasible for CTD because datasets contain divergent feature sets and model transferring can be a recommendation at best, and not a requirement. Similarly, [130] mention random sampling, but do not emphasize the statistical significance which is crucial for SsL in CTD due to the huge search space to extract a small L from a huge $\mathbb{X}$. This is less of a problem in e.g., Computer Vision, where most datasets (e.g., CIFAR) have been used thousands of times and benchmarks results are well-known; moreover, the corresponding community is more
open to source code disclosure. Because of these reasons, it is crucial to establish a specific set of requirements for CTD applications of SsL.

Regardless, some of our requirements are not met also by relevant works. An exemplary case, which exploits data augmentation in CIFAR, is MixMatch [20]. Here, no lower bound SL is considered: as a matter of fact, they only report the results of the upper bound SL trained on 50K samples (i.e., most of CIFAR). We do acknowledge that CIFAR is well-known and performance on small subsets can be easily assessed, but a fair comparison requires to evaluate such performance by using the same settings (i.e., same L and same classifier).

| Takeaway: the state-of-the-art does not allow to assess the benefits of SsL methods. Such immaturity is due to the lack of a rigorous evaluation protocol for SsL in CTD. |

5. Proposed Evaluation Framework

As a constructive step forward, we present CEF-SsL, an original Cybersecurity Evaluation Framework for SsL, which meets all the requirements in §3.3.

CEF-SsL aims to provide a practical assessment of SsL methods by using any fully labelled dataset, while simultaneously considering the deployment budget B and ensuring the statistical significance of the results. Because U can be easily obtained, CEF-SsL assumes that U is fixed and, hence, plays no role in practical comparisons. CEF-SsL has four inputs:

- D represents a large and fully labelled dataset. Such D can be either: (i) openly accessible; or (ii) created ad-hoc via simulations, well-known security feeds, or by manually labelling real data. CEF-SsL assumes that the labels in D are verified, i.e., all samples have the correct ground truth, which is the typical assumption in ML research. CEF-SsL uses D as basis to compose the four datasets required for a practical evaluation L, U, and F. CEF-SsL chooses samples from D uniformly at random. We observe that by setting different labelling costs C on each run, i.e., D', containing the performance on ‘future’ data; and F', containing any cost incurred during the development (not related to labelling).
- L is the labelling budget which is used to to compose L (cf Eq. 1). CEF-SsL assumes that L is fixed for the entire simulation. Variations of L imply different scenarios (hence, different L) which lead to unfair comparisons among models. If necessary, CEF-SsL can be applied again on different values of L.
- M denotes an array of ML methods. Such array must contain the specifics to devise all the baseline models (SL, SL, SsL) plus any additional model to be included in the evaluation. All the resulting models will be developed on the same labelling budget L.
- (n, k) is a pair of integers that regulate the ‘runs’ of CEF-SsL to achieve statistically significant results.

The output are two (n,k)-dimensional arrays, whose elements include the results of all the models devised through M on each run, i.e.: D', containing the performance on ‘future’ data; and F', containing any cost incurred during the development (not related to labelling).

We provide an overview of CEF-SsL in Fig. 4. CEF-SsL can be divided in three stages: Prepare, Run, Iterate.

5.1. Stage one: Prepare

The first stage uses L to partition D into L, U, and F. Fig. 5 shows a schematic of such a workflow.

- D is first split into F and U. Then L is further split into L according to L, and the leftout samples are considered as unlabelled U.

CEF-SsL begins by splitting D in F and L, the former, F, is used exclusively to assess the performance on future data; the latter, L is used for all remaining ‘training’ operations, because L can serve as basis to generate L, and then treat the remaining samples as unlabelled, representing U.

Generating F from D depends on the considered CTD task. The selection is done so as to achieve a representative |F| and ρ(F), while ensuring that the left-out samples (which will represent L) allow to create meaningful L and U. Such selection can also take into account the temporal relationships (if available) among the samples in D. For example, F can be composed by selecting only the ‘most recent’ samples in D, allowing assessment of potential concept drift [24].

To generate L from U, we recall that L = \sum_{x \in \mathbb{L}} Cx. Therefore, CEF-SsL chooses samples from L and assigns them to L, each time by decreasing the labelling budget L according to the cost of each sample14. However, L must include at least some benign and malicious samples. To this purpose, CEF-SsL requires a minimum amount of samples for each class. CEF-SsL will then populate L by randomly sampling as many samples of that class from L (while decreasing the labelling budget accordingly); then, CEF-SsL will use the remaining budget to further populate L randomly. We observe that by setting different labelling costs Cx it is possible to simulate imbalanced data distributions: we will showcase such intriguing property of CEF-SsL in our demonstration. At this point, CEF-SsL will consider all the samples in L not included in L as unlabelled, representing U. The case where one (or many) SsL approaches in M involve the usage of exploratory techniques is covered in the following stage.

5.2. Stage two: Run

The second stage begins with Reqs. 1–3. Specifically:

- 14. F does on depend on the labelling budget L and F \cap (U \cup L) = \emptyset.
- 15. Such cost can be fixed, or can vary depending on the desired level of realistic fidelity by associating each sample x \in D with a custom Cx.
1) train $SL$ on $L$ and test it on $F$ as $\mu(SL)$; account for all operational costs as $\varepsilon(SL)$;
2) train $SL$ on $L$ and test it on $F$ as $\mu(SL)$; account for all operational costs as $\varepsilon(SL)$;
3) use $L$ and $U$ to devise $SsL$, and test it on $F$ as $\mu(SsL)$; account for all operational costs as $\varepsilon(SsL)$.

Then, CEF-SsL focuses on each remaining method in $ML$:
- if the SsL method does not make any assumptions on $L$, then CEF-SsL uses $U$ and the previously drawn $L$ as input for the SsL method.
- if the SsL method requires $L$ to be composed in a more refined way, then CEF-SsL generates a new $L$ according to the specifics of the SsL method. The cost of labelling is ‘charged’ to $L$ and all other costs are accounted as $\varepsilon(SsL)$. The resulting $L$ and (new) $U$ are then used as input to the SsL method.
- Finally, the respective $SsL$ model is trained, and tested on $F$ resulting in $\mu(SsL)$; all the operational costs are accounted in $\varepsilon(SsL)$.

At the end of this stage, CEF-SsL populates the respective $\overline{\mu}$ and $\overline{\varepsilon}$ with the specific performance and (extra) costs of all the considered models of this single ‘run’.

### 5.3. Stage three: Iterate

To obtain statistically significant results, CEF-SsL performs multiple ‘runs’, according to the iteration parameters $(n,k)$ provided as input. Specifically:
- CEF-SsL repeats its entire workflow $k$ times, each time by choosing a new $F$, leading to different $L$ and, hence, different $L$ and $U$.
- For each (new) $F$, CEF-SsL composes a different $L$ (and, hence, different $L$) for $n$ times to account for randomness.

Altogether, CEF-SsL will evaluate each method in $ML$ a total of $n \times k$ times, resulting in as many $\mu$ and $\varepsilon$, all of which will be inserted into $\overline{\mu}$ and $\overline{\varepsilon}$. Assessment of different $L$ requires CEF-SsL to be restarted by providing a different $F$ while maintaining all other inputs.

Finally, the aggregated results of each method can be validated via, e.g., a Student t-test [131] or a more refined Wilcoxon Ranksum test [132]—provided that CEF-SsL is run enough times to provide statistically significant results (e.g., 50s or more [133]). Such comparisons can be made by considering different initial conditions (e.g., different $L$), because CEF-SsL ensures that all such conditions are shared across all methods; hence, the only difference is which ML method produced each single result.

### 6. Demonstration

As a final contribution of our paper, we demonstrate the application of our CEF-SsL framework to assess the benefits of SsL in CTD. We do so through a massive experimental campaign using 9 well-known datasets, in which we consider 9 existing SsL methods. We aim to:
- show how to apply our cost model and CEF-SsL;
- further motivate the importance of our requirements;
- provide the first statistically validated benchmark for future studies.

All our considered datasets are publicly available, and we release the code of our CEF-SsL implementation [16].

We first outline the chosen datasets (§6.1). Next, we explain the considered SsL methods (§6.2) and their implementation through CEF-SsL (§6.3). We then summarize the results (§6.4) and showcase a statistically significant comparison (§6.5). We report in the Appendix some low-level details (§A) and the full benchmark results (§B).

### 6.1. Datasets

Our evaluation focuses on the three CTD areas considered in this paper (§2): NID, PWD, MD. For each area we consider three publicly available datasets.

For NID, we use: CTU13, UNB15, IDS17. These datasets are well-known in the NID community, and contain data representing a mixture of synthetic and real network traffic captured in large environments. CTU13 is provided as PCAP traces and is focused on botnet; UNB15 and IDS17 are provided as NetFlows and contain additional malicious activities such as DoS, exploits, or illegitimate recon.

For PWD, we use: UCB, Phish, Mendeley. These well-known datasets contain webpage information, such as the URL, the reputation of the website, and the contents of the source HTML. Two (UCB and Mendeley) are provided directly as features, while Phish has raw webpages, from which we extract the features by following established practices [134].

For MD, we use: Drebin, Ember, AndMal20. These datasets are widely employed for ML-related analyses on malware targeting different OS: Ember for Windows, Drebin and AndMal20 for Android. Although Drebin is becoming outdated (it was collected in 2013), AndMal20 is very recent and serves for a better representation of current trends.

After obtaining all these 9 datasets, we preprocess them so that they are usable for our objectives. For instance, we clean some redundant data, or derive their feature sets. Some datasets may contain samples of different malicious classes. Because our focus is on detection, in some cases (Drebin, AndMal20) we aggregate all of them into a single malicious class. Datasets for PWD and Ember only have one malicious class. Datasets for NID have a huge variety: as done in previous work [40], we consider one classifier per each specific attack. More details are in the Appendix A.1. Regardless, all these operations are fixed for all the ML models evaluated on each dataset, meaning that their impact on $\varepsilon$ is the same for each model and, hence, negligible for comparisons.

We provide an overview of these datasets after all preprocessing has taken place in Table 2. For each dataset $D$, we report the overall amount of benign and malicious samples ($D_b$ and $D_m$, respectively), the amount of malicious classes ($N$), the reiterations $(n,k)$ performed by CEF-SsL, and reference to a past work that used such dataset (SotA).

For each dataset, CEF-SsL performs all experiments $N$ times, totalling $N \times (n \times k)$ runs.

### 6.2. Selected SsL Methods

Let us describe the SsL methods that are included in $ML$ in out study, together with the two SL baselines.

16. Available at: https://github.com/hihey54/CEF-SsL
We combine pseudo labelling with active learning by using \( \pi \text{SsL} \) as initial learner (but developed with half of the initial \( \mathcal{L} \)), which produces three ‘pseudo-active’ methods (\( \alpha \pi \text{SsL}_1 \), \( \pi \alpha \text{SsL}_n \), \( \alpha \pi \text{SsL}_n \)) in the same way as in the ‘pure’ active learning.

**Evaluation.** We evaluate all SsL methods proposed in the state-of-the-art (cf. Table 1) is clearly infeasible and also impossible due to their limited reproducibility and different assumptions. To showcase all scenarios envisioned in our CEF-SsL framework, we consider 9 SsL methods which are variations of two established SsL methods: self learning via pseudo-labelling (e.g., [65]) and active learning via uncertainty sampling (e.g., [66]), summarized in §2.3. Specifically, we consider 3 ‘pure’ pseudo-labelling methods, 3 ‘pure’ active learning methods, and 3 combinations thereof (e.g., [70]), where we cascade pseudo-labelling with active learning. The decision criterion is the confidence threshold \( c \).

**Pseudo Labelling.** One of the ‘pure’ pseudo labelling models represents our SsL baseline. We devise:

- \( \text{SsL} \), using all pseudo labels regardless of their confidence; the process is entirely automated because \( \mathcal{L} \) is chosen randomly and no selection of \( c \) is required.
- \( \pi \text{SsL} \), using only the pseudo labels with the highest confidence \( c \geq 99\% \);
- \( \pi \text{SsL} \), which repeats the previous operation another time. We use \( \pi \text{SsL} \) to predict the remaining \( \mathcal{U} \), and insert the corresponding pseudo-labelled samples with \( c \geq 99\% \) in the ‘mixed’ \( \mathcal{L} \).

**Active Learning.** For these methods, we assume that half of the labelling budget is used to develop the first learner, and then the remaining half is used to assign the correct label (specified in the source dataset) to those samples that meet a specific confidence threshold, \( c \). To avoid bias, the ‘actively labelled’ samples are chosen randomly among those that meet the criteria. Due to such randomness, we repeat the draw 5 times for each active learning method. Depending on \( c \), we devise:

- \( \alpha \pi \text{SsL}_1 \), using low confidence samples, \( c \leq 1\% \);
- \( \alpha \pi \text{SsL}_n \), using high confidence samples \( c \geq 99\% \);
- \( \alpha \pi \text{SsL}_n \), using the other samples \( 1\% < c < 99\% \);

We note that our implementation of active learning is fundamentally different and more realistic than the one adopted by Pendlebury et al. [86]; in Tesseract, the oracle assigns the correct label to all samples within a certain confidence; on the other hand, we simply use the learner to provide a set of samples to the oracle, who can only label as many samples as allowed by the remaining budget. This ensures that the provided \( \mathcal{L} \) is never exceeded, allowing for fair comparisons.

**Pseudo-Active Learning.** We combine pseudo labelling with active learning by using \( \pi \text{SsL} \) as initial learner (but developed with half of the initial \( \mathcal{L} \)), which produces three ‘pseudo-active’ methods (\( \alpha \pi \text{SsL}_1 \), \( \alpha \pi \text{SsL}_n \), \( \alpha \pi \text{SsL}_n \)) in the same way as in the ‘pure’ active learning.

### TABLE 2: Considered Datasets

| CTD | Name | \( D_0 \) | \( D_m \) | \( n \) | \( (n, b) \) | SoA |
|-----|------|--------|--------|-----|---------|----|
| NID | CT013 [44] | 19.5M | 444K | 6 | (11.3) | [135] |
|     | UNB15 [45] | 100K | 2.22M | 4 | (23.6) | [136] |
|     | ID317 [43] | 55K | 2.21M | 5 | (15.3) | [137] |
| PWD | UCI [138] | 4898 | 6157 | 1 | (20.5) | [139] |
|     | Mendeley [140] | 58K | 31K | 1 | (20.5) | [141] |
|     | Andriel [50] | 5510 | 1013 | 1 | (20.5) | [50] |
| MD  | Drebin [142] | 122K | 4022 | 1 | (20.5) | [143] |
|     | Ember [143] | 400K | 400K | 1 | (20.5) | [145] |
|     | AndMal20 [144] | 162K | 195K | 1 | (20.5) | [147] |

### 6.3. Implementation

In our implementation, CEF-SsL performs the same operations for all considered \( \mathcal{D} \) and \( \mathcal{M} \).

The first stage of CEF-SsL is the creation of \( \mathcal{F} \) from \( \mathcal{D} \). In our case, we do so by adopting the 80:20 split which is common in CTD (e.g., [40], [148], [149]). Specifically, CEF-SsL randomly chooses 20% of the malicious samples in \( \mathcal{D} \) and 20% of the benign samples in \( \mathcal{D} \), and puts them into \( \mathcal{F} \). The resulting samples are then considered as \( \mathcal{L} \).

To allow a comprehensive benchmark, we consider three scenarios where the cost of labelling each sample varies depending on their class—which serves to investigate different balance ratios \( \rho (L) \). Specifically:

- (balanced) \( \rho_m = \rho_b \); the number of benign samples matches that of malicious, \( \rho (L) = (50, 50) \);
- (unbalanced) \( \rho_m = 2\rho_b \); the number of benign samples is twice that of malicious, \( \rho (L) = (66, 33) \);
- (very unbalanced) \( \rho_m = 5\rho_b \); the number of benign samples is five times that of malicious, \( \rho (L) = (84, 16) \).

Where \( b \) and \( m \) denote a benign and malicious sample, respectively. For each cost scenario, we vary the allocated labelling budget \( L \) four times. We do so by regulating the minimal amount of \( \text{benign} \) samples, \( L_b \), to be included in \( L \). Hence, CEF-SsL composes \( L \) by first selecting \( L_b \) benign samples from \( \mathcal{L} \); then, CEF-SsL keeps populating \( L \) by choosing malicious samples from \( \mathcal{L} \) until the budget \( L \) is depleted, resulting in \( L_m \) malicious samples. The values of \( L \) and \( L_m \) depend on each CTD task, and are reported in Table 3. Because each combination of \( \mathcal{L} \) and \( C \) represents a different setting, we restart CEF-SsL at each change (hence, 12 times) to allow a fair comparison.

### TABLE 3: Composition of \( L \) for different \( C \) and \( L \). In all cases, \( |L|=|L_m|+|L_b| \), \( F=0.2|D| \), \( |D|=0.8|D| \), and \( |U|=|L|-|L| \).

| \( CTD \) | \( \text{CID} \) | \( \text{NIID} \) | \( \text{PWD} \) | \( \text{MD} \) |
|--------|----------|--------|--------|--------|
| balanced | \( \rho_m = \rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |
|         | \( \rho_m = 2\rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |
|         | \( \rho_m = 5\rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |
| unbalanced | \( \rho_m = \rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |
|         | \( \rho_m = 2\rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |
|         | \( \rho_m = 5\rho_b \) | \( 100 \) | \( 90 \) | \( 80 \) | \( 70 \) |

For these methods, we assume that half of the labelling budget is used to develop the first learner, and then the remaining half is used to assign the correct label (specified in the source dataset) to those samples that meet a specific confidence threshold, \( c \). To avoid bias, the ‘actively labelled’ samples are chosen randomly among those that meet the criteria. Due to such randomness, we repeat the draw 5 times for each active learning method. Depending on \( c \), we devise:

- \( \alpha \pi \text{SsL}_1 \), using low confidence samples, \( c \leq 1\% \);
- \( \alpha \pi \text{SsL}_n \), using high confidence samples \( c \geq 99\% \);
- \( \alpha \pi \text{SsL}_n \), using the other samples \( 1\% < c < 99\% \);

We observe that our choices of \( L \) result in \( L \) that are smaller than the testing set \( \mathcal{F} \), which is a good practice in CTD research [54]. Overall, our models are trained with as little as 40 labels (for PWD), and as high as 2400 labels (for NID) The resulting sets \( \mathcal{L} \) and \( \mathcal{U} \) are immediately used to train the lower bound \( SL \) and the upper bound \( SL \), both tested on \( \mathcal{F} \). Then, CEF-SsL uses the remaining sets according to each SsL method in \( M \). More detailed information on such development can be found in Appendix A.2.

### 6.4. Evaluation

We apply the described implementation of CEF-SsL on each dataset. The considered ML methods use the Random Forest (RF) learning algorithm, for three reasons. First, because RF are widely adopted by past work (e.g., [40], [86], [143], [145]), favoring comparisons. Second, since RF are known to provide an excellent tradeoff
between performance and computation time, and they can also be parallelized: to provide statistically significant results we must consider thousands of models, hence we favor algorithms that are fast to train. Third, because preliminary analyses confirmed the previous statement: we empirically found that RF achieve similar performance as other algorithms (e.g., neural networks) while requiring a fraction of the training time.\textsuperscript{17}

**Performance Assessment.** We choose the F1-score (a positive is a malicious sample) as performance metric, which is common in CTD.\textsuperscript{18} We report in Table 4 the average F1-score achieved by each method in each dataset, across all the different combinations of \(\mathcal{L} \) and \(\mathcal{C}\). Granular analyses and model-to-model comparisons can be made by looking at the full results in Appendix B.

| TABLE 4: Average F1-score of all methods. Boldface denotes the best SSL method on each \(\mathcal{C}\). Gray cells denote the best ‘pure’ pseudo labelling method, while dark-gray is for the best active learning method. |
|----------------|----------------|----------------|----------------|
| Method         | CTD            | MD             | UNB15          |
|                | CTU13          | Mend           | DREBIN         |
| NID            | 0.0310         | 0.1971         | 0.0030         |
| SL             | 0.0411         | 0.1894         | 0.0028         |
| SsL            | 0.8643         | 0.8779         | 0.8759         |
| \(\alpha SsL\) | 0.4357         | 0.4777         | 0.4975         |
| \(\epsilon SsL\) | 0.4714         | 0.4824         | 0.4714         |

The following insights can be drawn from Table 4. First, albeit almost counter-intuitive, using all pseudo-labels is the most effective among the ‘pure’ pseudo labelling techniques. Second, despite the identical labelling budgets in NID datasets, SSL methods achieved varying performance: in **\textbf{NID15}** they achieve only 0.6 F1-score at best, whereas in **\textbf{2021}** they can reach almost 0.9 F1-score; this highlights the importance of conducting evaluations on diverse datasets. Third, active learning appears to be the best way to use the labelling budget, but in **\textbf{Finch}** it is always inferior to ‘pure’ pseudo labelling; moreover, it is interesting that the results of the models trained on the (correct) high confidence labels consistently achieve the worst performance. Fourth, in all datasets, all models performed very similarly (on average) to the baseline **SL**. Such small gap requires to be further investigated via statistical comparisons, which we will do in \$6.5\$.

**Assessment of Extra Costs.** Many factors contribute to \(\epsilon\) for each considered ML method. All of our implemented methods share the same testbed, and most of such costs are equal for all methods. We hence focus on the most salient ‘cost’ of each method that we can measure: its execution time.

We report in Table 5 the total time required to develop each model, which comprises all the steps for (re)training and (for SSL methods) predicting unlabelled data.

| TABLE 5: Average execution time (seconds) of all methods. Bold values denote the SSL method with best F1-score on the same \(\mathcal{D}\) (cf. Table 4). |
|----------------|----------------|----------------|----------------|
| Method         | CTD            | MD             | UNB15          |
|                | CTU13          | Mend           | DREBIN         |
| NID            | 0.0310         | 0.1971         | 0.0030         |
| SL             | 0.0411         | 0.1894         | 0.0028         |
| SsL            | 0.8643         | 0.8779         | 0.8759         |
| \(\alpha SsL\) | 0.4357         | 0.4777         | 0.4975         |
| \(\epsilon SsL\) | 0.4714         | 0.4824         | 0.4714         |

In contrast, all other models are trained on a much smaller dataset. However, using only the execution time when comparing the \(\epsilon\) is not always fair. Some models may be better in terms of execution time but require some manual tuning, e.g., setting the desired level of confidence \(c\). Compare, for instance, **SsL** and **\(\alpha SsL\)** on **CTU13**: the former requires 35s, whereas the latter only 3s, which is a 32s difference. However, choosing an appropriate \(c\) for **\(\alpha SsL\)** requires: (i) inspecting the results of **\(\alpha SsL\)**, (ii) setting the new \(c\), (iii) devising a new **\(\alpha SsL\)**, (iv) inspecting the results of the new **\(\alpha SsL\)**, and (v) deciding whether it has acceptable performance or not. These procedures have a human in the loop and hence a significantly higher \(\epsilon\) (which cannot be shown in Table 5). On the other hand, **SsL always** achieves the reported performance, and by being entirely automated will result in an overall lower \(\epsilon\).

6.5. Statistical Validation

To substantiate the claim that some SSL using \(\bigcup\) outperforms the respective baseline (**SL** or **SsL**), statistical tests can be carried out. Here, we use the Wilcoxon ranksum [132], in which two populations are compared with the goal of verifying a given null hypothesis \(H_0\). The test outputs a \(z\)-value used to derive a \(p\)-value: \(H_0\) can be accepted or rejected on the basis of \(p\), according to a target significance level. We use the two-tailed version of the test, hence \(H_0\) is that the two populations are statistically equivalent; the larger the \(p\)-value, the more \(H_0\) should be accepted (and vice versa). We set the significance level to 0.05, implying that if \(p > 0.05\) then the two populations are equivalent; conversely, if \(p \leq 0.05\), the two populations are different (this is especially true if \(p \approx 0.05\)).

For each dataset, we compare the populations containing the performance of the baseline **SL** against: (i) the best ‘pure’ pseudo-labelling method (gray cells in Table 4); and (ii) the best active learning method (dark gray cells in Table 4). We are comparing\textsuperscript{19} the methods, not the individual models (model-to-model comparisons can be made from Figs. 6–8). The results of such tests are in Table 6, reporting the size of the populations, and the output \(z\) - and \(p\) values.\textsuperscript{21}

From Table 6, we can draw the following conclusions.

19. The test is valid: the compared populations have the same amount of elements and the conditions are shared among all elements, where the only difference is the generation process (i.e., the specific ML method).
20. Such size is given by \(n^k \times 12\), where we consider 3 cost scenarios with 4 budgets. For NID, each element is the average of the N malicious classes.
21. The EffectSize of the test can be derived by \(\sqrt{\frac{z^2}{n}}\).
TABLE 6: We statistically compare the SL baseline method against the best ‘pure’ pseudo-labelling and the best active learning methods. Bold values denote when $H_0$ is accepted ($p > 0.05$), i.e., the two methods are statistically equivalent. Cells in green (red) denote cases where using $U$ statistically increases (decreases) performance.

| Dataset | Purse | Best ‘pure’ pseudo-labels | Best active learning |
|---------|-------|---------------------------|----------------------|
| UCI     | 396   | 977                       | α_{SsL} > 0.001, 4.310 |
| Under   | 1164  | 977                       | α_{SsL} > 0.001, 15.98 |
| IDS1    | 546   | 977                       | α_{SsL} > 0.001, 0.05 |
| UB15    | 1200  | 977                       | α_{SsL} > 0.001, 0.05 |
| Nerd    | 1200  | 713                       | α_{SsL} > 0.001, 6.757 |
| Ember   | 1200  | 977                       | α_{SsL} > 0.001, 6.757 |
| AndMal20| 1200  | 713                       | α_{SsL} > 0.001, 6.757 |

Active Learning provides statistically significant improvements, which can be remarkable (cf. Figs 6–8); a finding that is consistent with past works (e.g., [95]). However, this is not always true: on IDS17, $\alpha_{SSL}$ is statistically equivalent to $SL$, meaning that there is no benefit in using $U$ on IDS17 (at least according to our testbed). Moreover, it can be detrimental: on Ember, the best method using active learning ($\alpha_{SSL}$) yields lower performance than the baseline $SL$, and such difference is statistically significant ($p=0.002<0.05$).

Pseudo Labelling is not useless. In UNB15, the pseudo-active method $\alpha_{SSSL}$ statistically outperforms the baselines ($SL$, and also $SSL$). However, in all its ‘pure’ applications, it provides no benefit: its best performer is $SSL$, which is always statistically equivalent to $SL$. To put it differently, in the ‘worst case’ using $U$ will not induce performance loss.

Finally, we also conducted the one-tailed variant of the statistical test, which confirms all previous findings.

7. Discussion and Future Work

We make some crucial remarks on our evaluation.

Importance of our Requirements. Without considering $SL$, it was not possible to determine that any ‘pure’ pseudo-labelling model was not just useless but even detrimental, due to achieving the same or inferior $\mu(\cdot)$ while requiring higher $B$. For instance, consider the detailed results on UCI in Fig. 7a: the $SSL$ models achieve a respectable F1-score of 0.9 when trained only on 40 labelled examples. However, the same result is achieved by the baseline $SL$, which does not make use of an $U$ containing 8000 samples. Similarly, on AndMal20, investing in $U$ provides very little benefit because the performance gap between $SL$ and $SSL$ is marginal. Furthermore, active learning was considered to be superior to random sampling [12], but for two of our datasets, IDS17 and Ember, this cannot be confirmed. Such insights would not have been possible without a massive and statistically validated comparison: looking only at Table 4, one may conclude that the best active learning model has better average performance than the $SL$ baseline. Finally, by considering Req. 2, it was possible to determine the lowest cost induced by using $U$ in the $SSL$ pipeline due to lack of human supervision (cf. Table 5).

Balancing. An intriguing occurrence can be observed from Figs. 6–8. In the presence of data-imbalance, the performance can be lower despite a higher labelling budget. This is evident for Ember (Fig. 8b). In the balanced scenario where $C_m=C_0$ (leftmost plot): when $L$ contains 800 correct labels, all models converge to a 0.8 F1-score; however, when $C_m=3C_0$ (rightmost plot), the performance when $L$ has 2400 correct labels ranges from 0.55 to 0.75 F1-score. We tried to regulate these situations by applying oversampling techniques (e.g., [150]), but we never saw significant changes. Such interesting result also occurs on AndMal20 and UCI despite at a lower magnitude.

Scope. Despite our extensive evaluation, we stress that our results should serve as a basis for future works, and should not be used to derive ‘universal’ statements. For instance, the considered SSL methods represent just a subset of all conceivable SSL techniques: hence our experiments cannot be used to conclude that “all pseudo-labelling techniques are not very effective in CTD”. In addition, we stress that we apply well-known methods on existing datasets and do not claim superiority over past works. In Appendix C, we present a case study comparing our results with a recent paper [70] using a similar testbed as ours. Furthermore, although our cost model (cf. §3.2) can represent any classification problem, in our evaluation we use CEF-SSL to assess binary classifiers. This is because the main goal of CTD is separating threats from legitimate events, but we acknowledge that some applications may involve fine-grained analyses. Evaluations of SSL methods in multi-classification settings are challenging, and we provide some considerations in Appendix D.

Labelling Accuracy. We assume that all the considered datasets are correctly labelled. However, such assumption may be overly optimistic: as described in §2, manual labelling is an error-prone task, and some recent papers highlighted that even well-known datasets may contain flaws (e.g., [151]). Due to the seminal nature of this SoK paper, we do not make any change to the provided labels—which also facilitates comparisons with previous works using such datasets, as the ground truth is the same. Nevertheless, we endorse future studies to question the correctness of the labelling procedures—especially if aimed at realistic deployments, as wrong labels induce data poisoning (which can also affect unlabelled samples [152]).

Concept Drift. Our evaluation assumes that the data distribution is stationary. In reality evaluations should be performed at regular intervals to prevent concept drift (cf. §3.2). Such operations, however, are facilitated by the design of our framework: CEF-SSL can also be applied to lifelong learning scenarios by selecting $F$ and $L$ on a temporal basis. For instance, in Tesseract [86], the initial model is developed under the assumption that all samples from the past are correctly labelled, resulting in over 50K samples. Our CEF-SSL can be applied by assuming that only a fraction of ‘past’ samples are correctly labelled ($L$) while the remaining ones are not verified ($U$), and the most recent samples are used as test ($F$).

8. Conclusions

Acquiring ground truth information in CTD is difficult, but large amounts of unlabelled data are regularly available. These premises make SSL an intriguing opportunity.

22. We will update our GitHub repository with new findings if such findings are derived from CEF-SSL.
as it exploits unlabelled data to mitigate the problem of scarce ground truth. While many previous works have employed SsL for diverse CTD tasks, none of them investigated the benefit provided by unlabelled data. Despite being relatively cheap, such data still brings certain costs into the ML pipeline.

In this SoK paper, we specifically investigate the utility of unlabelled data and hence facilitate deployment of SsL methods for CTD. We formalize the evaluation requirements that enable one to assess the impact of unlabelled data in the development of a SsL model, under the assumption of a limited labelling budget. Prior works in this area had different scope and never considered such requirements, hence the impact of unlabelled data could not be assessed.

As a constructive demonstration, we provide an evaluation framework that meets all these requirements and use it to perform the first statistically validated benchmark of 9 selected SsL methods on 9 well-known datasets for CTD. The results reveal that only SsL methods using active learning are statistically better than baselines that do not use unlabelled data; however, in some cases they can degrade performance.

Our paper hence highlights the substantial margin for improvement of SsL methods for CTD. This motivates the quest for future contributions that exploit unlabelled data in CTD, compensating the high cost of expert knowledge in this field.

References

[1] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” Science, vol. 349, no. 6245, pp. 255–260, 2015.
[2] S. M. Ghaffarian and H. R. Shahriari, “Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey,” ACM Comp. Surv., vol. 50, no. 4, pp. 1–36, 2017.
[3] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian, “Darknet and deepnet mining for proactive cybersecurity threat intelligence,” in Proc. IEEE Conf. Secur. Inf., 2016, pp. 7–12.
[4] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, “On the effectiveness of machine and deep learning for cybersecurity,” in Proc. IEEE Int. Conf. Cyber Conflicts, May 2018, pp. 371–380.
[5] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Proc. IEEE Symp. Secur. Privacy, 2010, pp. 305–316.
[6] B. Miller, A. Kanchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullahbo, L. Huang, V. Shankar, T. Wu, G. Yu et al., “Reviewer integration and performance measurement for malware detection,” in Proc. Int. Conf. DIMVA, 2016, pp. 122–141.
[7] R. Meier, C. Scherrer, D. Gugelmann, V. Lenders, and L. Vanbever, “Feedrank: A tamper-resistant method for the ranking of cyber threat intelligence feeds,” in Proc. IEEE Int. Conf. Cyber Conflict, 2018.
[8] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, “Scalable, behavior-based malware clustering,” in Proc. Netw. Distrib. Syst. Secur. Symp., vol. 9, 2009.
[9] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, “Ec2: ensemble clustering and classification for predicting android malware families,” IEEE Trans. Depend. Sec. Comput., 2017.
[10] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” Machine Learning, vol. 109, no. 2, pp. 373–440, 2020.
[11] J. Koza, M. Krcal, and M. Holena, “Two semi-supervised approaches to malware detection with neural networks,” in ITAT, 2020, pp. 176–183.
X.-Y. Zhang, S. Wang, X. Zhu, X. Yun, G. Wu, and Y. Wang, “Detecting unknown computer worm activity via support vector machines and active learning,” Pattern Anal. Appl., vol. 15, no. 4, pp. 459–472, 2012.

M. Zhao, T. Zhang, F. Ge, and Z. Yuan, “RoboDroid: a lightweight malware detection framework on smartphones,” J. Netw., 2016.

N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, “Novel active learning methods for enhanced PC malware detection in windows OS,” Elsevier Exp. Syst. Appl., vol. 41, no. 13, pp. 5843–5857, 2014.

X.-Y. Zhang, S. Wang, X. Zhu, X. Yun, G. Wu, and Y. Wang, “Update vs. Upgrade: modeling with indeterminate multi-class active learning,” Elsevier Neurocomput., vol. 162, pp. 163–170, 2015.

N. Nissim, R. Moskovitch, O. Barad, L. Rokach, and Y. Elovici, “ALDROID: efficient update of Android anti-virus software using designated active learning methods,” Springer Knowl. Inf. Syst., vol. 49, no. 3, pp. 795–833, 2016.

M. Ni, T. Li, Q. Li, H. Zhang, and Y. Ye, “FindiMal: A file-to-file social network based malware detection framework,” Elsevier Knowl. Based Syst., vol. 112, pp. 142–151, 2016.

L. Chen, M. Zhang, C.-Y. Yang, and R. Sahita, “POSTER: Semi-supervised classification for dynamic Android malware detection,” in Proc. ACM Conf. Comp. Commun. Secur., 2017, pp. 2479–2481.

Y. Fu and J. Xu, “Malware detection via extended label propagation through graph inference,” IEEE Access, vol. 7, 2019.

P. Irofti and A. Băltăiu, “Malware identification with dictionary learning,” in Proc. IEEE Eusipco, 2019, pp. 1–5.

S. Sharmeen, S. Huda, J. Abawajy, and M. M. Hassan, “An adaptive framework against android private elevation threats using deep learning and semi-supervised approaches,” Elsevier Appl. Soft Comp., vol. 89, p. 106089, 2020.

C.-W. Chen, C.-H. Su, K.-W. Lee, and P.-H. Bair, “Malware family classification using active learning by learning,” in IEEE ICACT, 2020.

Q. Li, Q. Hu, Y. Qi, S. Qi, X. Liu, and P. Gao, “Semi-supervised two-phase familial analysis of android malware with normalized graph embedding,” Knowledge-Based Systems, vol. 218, p. 106802, 2021.

M. Hutson, “Artificial Intelligence Faces Reproducibility Crisis,” Science, vol. 359, no. 6377, pp. 725–726, 2018.

M. Dehghani, Y. Tay, A. A. Gritsenko, Z. Zhao, N. Houlsby, F. Diaz, D. Metzler, and O. Vinyals, “The benchmark lottery,” in NeurIPS, 2021.

K. Musgrave, S. Belongie, and S.-N. Lim, “A metric learning reality check,” in Proc. Europ. Conf. Comp. Vis., 2020, pp. 681–699.

D. W. Zimmerman, “Comparative power of student t test and mann-whitney u test for unequal sample sizes and variances,” The Journal of Experimental Education, vol. 55, no. 3, pp. 171–174, 1987.

S. Datta and G. A. Satten, “Rank-sum tests for clustered data,” J. Amer. Statist. Assoc., vol. 100, no. 471, pp. 908–915, 2005.

M. Hopp, A. C. Bathke, and E. Brunner, “Optimal sample size planning for the wilcoxon-mann-whitney test,” Statistics in medicine, vol. 38, no. 3, pp. 363–375, 2019.

R. M. Mohammad, F. Thabtah, and L. McCluskey, “Intelligent rule-based phishing websites classification,” IET Inf. Secur., 2014.

G. Apruzzese, M. Colajanni, and M. Marchetti, “Evaluating the effectiveness of adversarial attacks against botnet detectors,” in Proc. IEEE Int. Symp. Netw. Comput. Appl., Oct. 2019, pp. 1–8.

S. Rajagopal, P. P. Kundapar, and K. S. Harresha, “A stacking ensemble for network intrusion detection using heterogeneous datasets,” Secur. Commun. Netw., vol. 2020, 2020.

M. Di Mauro, G. Galatro, and A. Liotta, “Experimental review of neural-based approaches for network intrusion management,” IEEE T. Netw. Serv. Manag., 2020.

M. Karabatak and T. Mustafa, “Performance comparison of classifiers on reduced phishing website dataset,” in Proc. IEEE Int. Symp. Digit. Forensic Secur., 2018, pp. 1–5.

S. R. Sharma, R. Parthasarathy, and P. B. Honnavalli, “A feature selection comparative study for phishing datasets,” in Proc. IEEE Int. Conf. Elec. Comp. Commun. Tech., 2020, pp. 1–6.

G. Vrbančić, “Phishing Websites Dataset—Mendeley Data,” 2020.

M. Al-Sarem, F. Saeed, Z. G. Al-Mekhlafi, B. A. Mohammed, T. Al-Hadhrami, M. T. Alshammari, A. Alreshidi, and T. S. Al-shammari, “An optimized stacked ensemble model for phishing websites detection,” Electronics, vol. 10, no. 11, p. 1285, 2021.

D. Arp, M. Spreitenbarth, H. Gascon, K. Reck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket,” in Proc. Netw. Distrib. Syst. Secur. Symp., 2014.

M. S. Rana, C. Noulas, and A. H. Sung, “Evaluating machine learning models for android malware detection: A comparison study,” in Proc. Int. Conf. Netw. Commun. Comp., 2018, pp. 17–21.

H. S. Anderson and P. Roth, “Embcr: an open dataset for training static pe malware machine learning models,” arXiv:1804.04637, 2018.

C. Galen and R. Steele, “Empirical measurement of performance maintenance of gradient boosted decision tree models for malware detection,” in Proc. IEEE Int. Conf. Artif. Int. Inf. Commun., 2021.

A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, “DiDroid: Android Malware Classification and Characterization Using Deep Image Learning,” in Proc. Int. Conf. Commun. Netw. Secur., 2020, pp. 70–82.

D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon, and F. Massicotte, “EntropLyzer: Android Malware Classification and Characterization Using Entropy Analysis of Dynamic Characteristics,” in Proc. IEEE RDAAPS, 2021, pp. 1–12.

H. Alshahrani, H. Mansourt, S. Thorn, A. Alshehi, A. Alzahrani, and H. Fu, “iDefender: Android application threat detection using static and dynamic analysis,” in Proc. Int. Conf. Consum. Elec., 2018.

A. Awasthi and N. Goel, “Phishing website prediction: A machine learning approach,” in Progress in Advanced Computing and Intelligent Engineering. Springer, 2021, pp. 143–152.

S. Barua, M. M. Islam, X. Yao, and K. Murase, “MMOTe-majority weighted minority oversampling technique for imbalanced data set learning,” IEEE T. Knowl. Data Eng., vol. 26, no. 2, 2012.

G. Engelen, V. Rimmer, and W. Joosen, “Troubleshooting an intrusion detection dataset: the CICIDS2017 case study,” in Proc. IEEE Secur. Privacy Workshops, 2021, pp. 7–12.

N. Carlini, “Poisoning the unlabeled dataset of semi-supervised learning,” USENIX Secur. Symp., 2021.

G. Apruzzese, M. Andreolini, M. Colajanni, and M. Marchetti, “Hardening Random Forest Cyber Detectors Against Adversarial Attacks,” IEEE T. Emerg. Topics Intell. Syst., vol. 4, no. 4, pp. 427–439, 2020.

R. Panigrahi and S. Borah, “A detailed analysis of cicids2017 dataset for designing intrusion detection systems,” Int. J. Eng. Techn., 2018.

Appendix A.

Experimental Testbed

All our experiments are performed on a machine equipped with an Intel Xeon W-2195 CPU with 36 cores, 256GB RAM, 2TB SSD NVMe, and Nvidia Titan RTX GPU. The implementation leverages Python3 and the well-known ML library of scikit-learn. The specific ML algorithm used as base for all our models is the RandomForestClassifier. Training such classifier can be parallelized: in particular, we set the njobs parameter to use 34 cores (out of 36) of our CPU.
A.1. Data Preprocessing

**NID datasets.** The considered NID datasets all contain more than one malicious class—which we treat differently.

- **ctu13** is composed of 13 PCAP traces, each one containing either benign or malicious traffic belonging to a given botnet family out of 7 possible families. Such traces are transformed into NetFlows via Argus. To prevent overfitting, we remove the full IP-address (we differentiate between internal/external IPs); we also derive some additional metrics (e.g., bytes per second, or the IANA port categories). The entire feature set is provided in Table 7, which is similar to the one in [153]. Then, we merge all sets according to the specific botnet family, obtaining 7 sets containing either benign samples, or malicious samples belonging to a single family. We exclude those botnet families with less than 1K samples.

- **ids17** is provided as NetFlows, each separated in benign and malicious samples of different attacks. We use the entire feature set provided by the authors of **ids17**, but we apply the same internal/external differentiation as done for **ctu13** to prevent overfitting on individual IP addresses. We aggregate the sets of NetFlows containing DoS attacks into a single set, and we exclude the families underrepresented. We create a dedicated set for the benign samples, which we use for all the experiments.

- **unb15** the procedure is similar to **ids17**. We use the same feature sets provided by the authors (with the usual distinction between internal/external machines). We aggregate the most represented families in 5 distinct sets.

**TABLE 7: Features considered for **CTU13**.**

| # Feature Name | Type |
|----------------|------|
| 1 SrcIP | Cat |
| 2 DstIP internal | Bool |
| 3 SrcPort type | Cat |
| 4 DstPort type | Cat |
| 5 SrcPort | Cat |
| 6 DstPort | Cat |
| 7 Flow Duration | Num |
| 8 Flow Direction | Bool |
| 9 Inc-Bytes | Num |
| 10 Out-Bytes | Num |
| 11 Total Bytes | Num |
| 12 Incoming Packets | Num |
| 13 Outgoing Packets | Num |
| 14 Total Packets | Num |
| 15 Bytes per Packet | Num |
| 16 Bytes per Second | Num |
| 17 Packets per Second | Num |
| 18 Inc- out-Bytes | Num |

The complete break down of the benign and malicious sets for each NID dataset is provided in Table 8.

**TABLE 8: Breakdown of traces for NID datasets.**

| Dataset | Trace | Attack Class | Malicious Samples | Benign Samples |
|---------|-------|--------------|-------------------|----------------|
| ctu13   |       |              |                   |                |
| ids17   |       |              |                   |                |

In all experiments involving NID datasets, we devise ensemble of classifiers, each focused on a specific family (as done in [70], [153]). The results presented in our graphs are the average of all classifiers.

**MD datasets.** The task of MD has been tackled either as a binary classification [145] or as a multi-classification ML problem [9]—the latter focusing on identifying the specific malware family of a given sample. Because **EMBER** only has a single malicious family, all our MD are binary classifiers. To this purpose, we observe that **DREBIN** (collected from 2010 to 2012) has hundreds of malware families, some of which extremely underrepresented: hence, we considered the samples of top10 families and treated them as a single class. On the other hand, **AndMal20** is more balanced across families, hence we consider all samples as a single malicious class.

**P WD datasets.** All P WD datasets include malicious samples of a single class. To the best of our knowledge, P WD has always been treated as a binary classification problem; hence, developing our P WD is more straightforward compared to NID and MD. The **UCI** and Mendeley datasets are provided directly as features; whereas for **Phish**, we extract the most significant features based on the information of each provided webpage (HTML, URL, DNS) to achieve a similar feature set as the other two [53]. The queries to the DNS servers were performed in 2019, and some returned null results; we still consider these pages in our evaluation, as this happened for both benign and malicious pages. We provide in Table 9 the complete feature set used for **Phish**.

**TABLE 9: Features computed for the **Phish** dataset.**

| Feature | Type     |
|---------|----------|
| IP address | URI | DNS-byte
| URL-features | Fak | Dots number
| | | Shortener
| DataURI | Redirect |
| | | Symbol

A.2. Developing the SSL models

We describe our SSL methods and provide an example. ‘Pure’ pseudo-labelling. We first use **SL** to predict the pseudo-labels of **U** and provide the confidence, **c** of such predictions. Depending on **c** (see §6.2), the pseudo-labelled samples will be inserted in **L**, resulting in a ‘mixed’ **L** used to train a ‘pure’ pseudo-labelling model. For the pseudo-labelling with retraining, we use **π SSL** to predict the labels of the remaining samples in **U**, and assign those with **c ≥ 99%** to **L** (with the pseudo-label).

‘Pure’ active learning. CEF-SSL composes another **L** (and another **U**) with a two-step approach, by assuming that half of the labelling budget is used initially, and the remaining half is used for labelling the suggested samples. To this purpose, the initial **L** is changed by randomly removing half of its benign and malicious samples, therefore restoring the budget **L** to half of its initial value. Such **L** is used to train a ‘support’ **SL** model that predicts the labels of **U** and the corresponding confidence, **c**. CEF-SSL then simulates a human oracle that assigns the correct label to the samples that meet a confidence threshold (cf.
§6.2) by accounting for such costs from the residual $L$. Such samples, with their correct label, will be inserted in $L$. To allow a fair comparison where all the models use an $L$ with the same size, we assume that the cost for labelling each ‘suggested’ sample is standardized. This assumption is realistic, because it assumes that the sample already exists (it comes from $U$) and a human operator can more efficiently verify a sample that is being provided with, compared to choosing and verifying samples randomly, or entirely creating new ones. The resulting new $\mathbb{L}$ (containing only correct labels) is then used to train the corresponding active learning model.

**Pseudo-active models.** We use the ‘support’ SL (trained on the ‘halved’ $L$) to predict the pseudo labels of $U$, and put all those with $\hat{c} \geq 99\%$ in $\mathbb{L}$ (with the pseudo label). Such ‘mixed’ $\mathbb{L}$ is used to train a support $\pi SsL$, which predicts the remaining $U$: the samples whose confidence meets the criteria in §6.2 are randomly put into the ‘mixed’ $\mathbb{L}$ until finishing the leftover budget. Such $\mathbb{L}$ (having the initial correct labels, the pseudo labels, and the ‘suggested’ correct labels) is used for training the pseudo-active SsL model.

**Example.** Consider the unbalanced case on CTU13 where $C_o=2C_b$, and $L=200$ (cf. Table 3). For the ‘pure’ pseudo models and the baseline SL, the $\mathbb{L}$ is used all at the beginning, and the final $L$ will always have 150 correctly labelled samples: 50 malicious samples and 100 benign. For active learning methods, the first half of the labelling budget (100) is used for the initial learner, by randomly choosing and removing 50 benign samples and 25 malicious samples from the previously randomly drawn $\mathbb{L}$; doing so leads to a smaller initial $L$ with 75 samples. Then, because of our standardized assumption, the oracle will randomly assign the correct label to 75 samples that meet the desired confidence criteria, irrespective of their class. The oracle will not label more samples than what it is allowed by the budget. Nonetheless, the corresponding model will be tested on $\mathbb{F}$, and then the entire process is repeated 5 times to account for randomness in choosing the ‘suggested’ samples.

**Appendix B. Benchmark**

We present our benchmark evaluation, whose nature is exploratory: we are not interested in providing results that ‘outperform’ the state-of-the-art. Our focus is providing the first statistically validated benchmark for SsL methods in CTD and promote future analyses. Hence, we consider realistic scenarios where the amount of labelled data is scarce: in our evaluation, we never use more than 2.4K labelled samples. As a consequence, some results may appear to be underwhelming: we consider such outcomes to be positive, as they highlight the huge improvement margin concealed by SsL.

**Results.** The overall benchmark results are shown in Figs. 6 for NID datasets; Figs 7 for PWD datasets; and Figs 8 for MD datasets. Each figure consists in a set of 3 subfigures, each focused on a specific dataset. Every sub-figure reports 3 plots, each focused on a specific ‘balance’ scenario (i.e., a specific $C$). Every plot reports the F1-score (vertical axis) achieved by all the considered SsL methods (lines) for increasing labelling budgets $L$ (horizontal axis). We observe that the performance of our MD for DREBIN (Fig. 8a) is significantly worse than on Ember (Fig. 8b) and on AndMal120 (Fig. 8c). Such phenomenon is due to our chosen aggregation strategy. Indeed, the considered malicious samples in DREBIN belong to 10 different families; however, such samples are randomly chosen when composing the $L$, i.e., $\mathbb{L}$ (which is very small) may not contain some families. As such, if these families are notably different from those included in $\mathbb{F}$ and—at the same time—vastly present in $\mathbb{F}$, then the MD will exhibit low performance. This phenomenon, however, does not appear in AndMal120 (for which we also aggregate families into a single class): an explanation is that the malicious families in AndMal120 are more similar to each other than those in DREBIN.

How many experiments does the benchmark include? We report in Table 10 the overall number of models considered in our evaluation. Specifically, for each dataset we report $n, k$ and the corresponding $N$. Multiplying all of these numbers yields the ‘runs’ of any model that does not leverage active learning, i.e., $\pi SsL, \pi SsL$, $\pi SsL$. Because we draw the samples for active learning randomly and we repeat such draw 5 times for each corresponding model, all the 6 active learning models (i.e., $\alpha SsL_b, \alpha SsL_o, \alpha SsL$, $\alpha SsL_b, \alpha SsL_o, \alpha SsL$) are assessed 5 times as much. Hence, for every line in a given plot, each ‘point’ is the average results of as many models as reported in Table 10. As an example, on CTU13, each point in a given plot of Fig. 6a is the average F1-score of 990 models if the line is related to an active learning method, or 198 models if not—all of which evaluated for the corresponding value of $L$ and cost scenario $C$.

**Appendix C. Case Study: Comparison with a prior work**

Let us discuss a case study where we compare our evaluation with a recent work sharing a similar testbed.

The combination of pseudo labelling and active learning has been investigated also by Zhang et al. [70] on CTU13 and IDS17 by using—among others—the same confidence levels as our implementation: above 99% for the pseudo labelling, and below 1% for active learning (making it equivalent to our $\pi SsL$ and $\alpha SsL$). In [70], the two source datasets (IDS17 and CTU13) are divided into a single set of benign samples; whereas the malicious samples are distributed into several sets depending on their attack.

| Dataset | Subfigure | $n$ | $k$ | $N$ | Active models ($\pi$) | Other models ($\pi$) |
|---------|------------|----|----|----|----------------------|---------------------|
| NID     | (Fig. 6a)  | 500 | 760| 99%| 100                  | 100                 |
| PWD     | (Fig. 7a)  | 460 | 1000| 99%| 100                  | 100                 |
| MD      | (Fig. 8a)  | 460 | 1000| 99%| 100                  | 100                 |

All the values in Table 10 correspond only to a specific combination of $L$ and $C$, meaning that the overall amount of models developed in our evaluation is 12 times as much—for each $\pi$. In summary, the results of SoK paper correspond to 500 760 active learning models and 83 460 non-active learning models, for a total of 584 220 models.
family, and then aggregated into a single malicious class. This is a valid operation, but it slightly differs from our testbed, because we treat malicious classes separately for each NID dataset.

Let us describe the evaluation methodology of [70]. Our first observation is that, after ‘preprocessing’ the source datasets, the authors of [70] obtain 100K samples for CTD and 600K samples for IDS17. In contrast, after preprocessing, we obtain 20M samples for CTD and 3M for IDS17—and this is despite removing some underrepresented families (cf. Table 2). We are not aware of the reason of this gap, but also other studies (e.g., [153], [154]) obtain similar compositions as ours.

Regardless, each preprocessed dataset in [70] is further filtered to obtain a D containing 15K benign samples and 9K malicious samples. Such D (which represents only a small portion of their preprocessed traffic, i.e. 10% for CTD, and 4% for IDS17) is never changed in the evaluation and the remaining samples are never used. In contrast, we consider as D the entire datasets after preprocessing.

Then, the authors of [70] partition such D into L and F by using a 70:30 split (we use 80:20), resulting in a L with 17K samples, and a F with 7K samples—both having ρ=[62,38]. The split is done randomly, but the process is never repeated and the F stays the same for the entire evaluation. What would have happened if F contained different samples? We address this issue by changing F multiple times (k) for each malicious class (N), meaning 18 times for CTD, and 15 times for IDS17.

To obtain their L (and corresponding U), the authors of [70] isolate a variable portion of samples from L, specifically either 5%, 10% or 20% (hence, the correct labels in their L range from ~1K to ~4K). The choice of samples put in L is done randomly in [70], but the experiments are repeated only 10 times. In contrast, we do so n times for each new F, meaning that we do so 198 times for CTD and 225 times for IDS17. This increases the confidence of our results. Moreover, we also consider different balance ratios, whereas the balancing in [70] is always a fixed ρ=[62,38] for all sets.

Finally, in [70] they consider the SL baseline, but neglect the SsL baseline and the SL baseline. Both lacks are significant: without SsL it is not possible to estimate the least possible benefit provided by U (if any); without SL it is not possible to determine any upper bound in performance. If the SL is not far from SL, then it may not be worth in investing in U for using SsL methods.

Let us compare our results, with the aim of pinpointing what ‘issues’ prevent deriving actionable conclusions on the impact of unlabelled data in [70]. For simplicity, we focus on the F1-score achieved on CTD (cf. Fig.6a).

The baseline SL in [70] achieves a lower performance than ours despite being trained on more samples: the one in [70] obtains 0.81 F1-score when trained on 4000 samples, whereas our baseline SL trained with an L of 2400 samples (the highest we considered) reached 0.87 F1-score, as evidenced by the rightmost plot in Fig.6a. When applying pseudo-labelling in [70], the performance increases from 0.81 to 0.83 F1-score. All these results contrast with ours, because the performance of our corresponding model, πSsL, is lower than the baseline. However, while our results take into account a total of 198 trials, the ones in [70] are performed only 10 times, which is hardly enough to make any informed decision on whether it is truly convenient to invest in U.

Finally, the authors of [70] do not maintain the original L when applying active learning, and do not report how many samples require to be ‘actively labelled’ as they inject all those within the confidence threshold (below 1%) into L (as also done in Tesseract [86]), making any comparison with our models (and, also, with their baselines) unfair.

We can conclude that the evaluation performed in [70] can only show that, under certain conditions, some SsL methods can improve the performance. However, the results obtained cannot certify that such improvement is significant, because they are conducted in a fixed setting (same F, same balance ratio, only 10 runs), despite being performed on two datasets. Hence, the question “do I need U?” is still open.

Appendix D.

CEF-SsL for multi-classification

In our evaluation we treat CTD as a binary classification problem, where a sample is either benign or malicious. The motivation is that N+1 classification (with N>1) assume a ‘closed world’ scenario where all malicious classes are known beforehand, which is hardly the case in the dynamic cybersecurity landscape. Nevertheless, some specific applications may favor SsL method devoted to multi-classification: applying CEF-SsL in similar settings is possible, i.e., by manually specifying the cost to label each malicious sample Cx, and performing hundreds of runs of CEF-SsL—but this is empirically difficult from a research perspective. Some alternatives exist, but they are also challenging.

The main challenge is randomly choosing a limited amount of samples from N malicious classes. For instance, some datasets (e.g., DREBIN) only have a very limited number of samples for some classes; a possible workaround is applying some aggregation techniques and create some ‘macro’ classes; however, such approach may introduce some experimental bias. Another option is removing those classes for which only few samples are available: in this case, however, the ML model may perform poorly if such families ‘appear’ after the model is deployed. To mitigate such problem, all underrepresented classes can be merged into a dedicated ‘other’ class: the ML model may retain some performance at inference; but it may also be confused when the more represented classes present similarities with the samples of the ‘other’ class.

Another challenge is composing ‘appropriate’ partitions, i.e., L, U and F. It is well-known that, in reality, benign events are more abundant than malicious ones, and it is common to compose train/test partitions where benign samples are the majority. However, when malicious samples belong to different classes and the labeling budget is limited, it begs the question of “how many samples per class should be included in L?” As an example, assume that L=500; how should L be composed when a dataset contains 10K benign samples, alongside three malicious classes, the first with 2000 samples, the second with 100 samples, and the third with 50 samples? And what about U and F? A possibility is using the relative distribution in
(a) Results on CTU13. For every plot, each ‘point’ represents the average results of 198 models (and 5 times as many for all models using active learning).

(b) Results on UNB15. For every plot, each ‘point’ represents the average results of 368 models (and 5 times as much for all models using active learning).

(c) Results on IDS17. For every plot, each ‘point’ represents the average results of 225 models (and 5 times as much for all models using active learning).

Figure 6: Network Intrusion Detection. For each dataset, we report the results for the three cost (or balancing) scenarios. Each scenario is shown in a plot, where the y-axis reports the F1-score and the x-axis the (increasing) labelling budget. Each method is denoted with a line on each plot.

...a given dataset, but it may induce bias or skew the model into favoring the majority class. Conversely, it is possible to infer which family is more popular ‘in the wild’, but this may require extra resources to study updated security feeds.
(a) Results on UCI. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

(b) Results on Mendeley. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

(c) Results on δPhish. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

Figure 7: Phishing Website Detection. For each dataset, we report the results for the three cost (or balancing) scenarios. Each scenario is shown in a plot, where the y-axis reports the F1-score and the x-axis the (increasing) labelling budget. Each method is denoted with a line on each plot.
(a) Results on DREBIN. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

(b) Results on Ember. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

(c) Results on AndMal20. For every plot, each ‘point’ represents the average results of 100 models (and 5 times as much for all models using active learning).

Figure 8: Malware Detection. For each dataset, we report the results for the three cost (or balancing) scenarios. Each scenario is shown in a plot, where the y-axis reports the F1-score and the x-axis the (increasing) labelling budget. Each method is denoted with a line on each plot.