AI/ML for Network Security: The Emperor has no Clothes

Extended Version — https://trusteeml.github.io/

Arthur S. Jacobs
UFRGS, Brazil
asjacobs@inf.ufrgs.br

Roman Beltiukov
UCSB, USA
rbeltiukov@ucsb.edu

Walter Willinger
NIKSUN Inc., USA
wwillinger@niksun.com

Ronaldo A. Ferreira
UFMS, Brazil
raf@facom.ufms.br

Arpit Gupta
UCSB, USA
arpitgupta@ucsb.edu

Lisandro Z. Granville
UFRGS, Brazil
granville@inf.ufrgs.br

ABSTRACT

Several recent research efforts have proposed Machine Learning (ML)-based solutions that can detect complex patterns in network traffic for a wide range of network security problems. However, without understanding how these black-box models are making their decisions, network operators are reluctant to trust them and deploy them in their production settings. One key reason for this reluctance is that these models are prone to the problem of underspecification, defined here as the failure to specify a model in adequate detail. Not unique to the network security domain, this problem manifests itself in ML models that exhibit unexpectedly poor behavior when deployed in real-world settings and has prompted growing interest in developing interpretable ML solutions (*e.g.,* decision trees) for "explaining" to humans how a given black-box model makes its decisions. However, synthesizing such explainable models that capture a given black-box model's decisions with high fidelity while also being practical (*i.e.,* small enough in size for humans to comprehend) is challenging.

In this paper, we focus on synthesizing high-fidelity and low-complexity decision trees to help network operators determine if their ML models suffer from the problem of underspecification. To this end, we develop TRUSTEE, a framework that takes an existing ML model and training dataset as input and generates a high-fidelity, easy-to-interpret decision tree and associated trust report as output. Using published ML models that are fully reproducible, we show how practitioners can use TRUSTEE to identify three common instances of model underspecification; *i.e.,* evidence of shortcut learning, presence of spurious correlations, and vulnerability to out-of-distribution samples.

1 INTRODUCTION

In the last few years, we have witnessed a growing tension in the network-security community. Recent research has demonstrated the benefits of Artificial Intelligence (AI) and Machine Learning (ML) models over simpler rule-based heuristics in identifying complex network traffic patterns for a wide range of network security problems (see recent survey articles such as [9, 47, 56, 63]). At the same time, we have seen reluctance among network security researchers and practitioners when it comes to adopting these ML-based research artifacts in production settings (*e.g.,* see [2, 4, 59]). The black-box nature of most of these proposed solutions is the primary reason for this cautionary attitude and overall hesitance. More concretely, the inability to explain how and why these models make their decisions renders them a hard sell compared to existing simpler but typically less effective rule-based approaches.

This tension is not unique to network security problems but applies more generally to any learning models, especially when their decision-making can have serious societal implications (*e.g.,* healthcare, credit rating, job applications, criminal justice system, etc.).

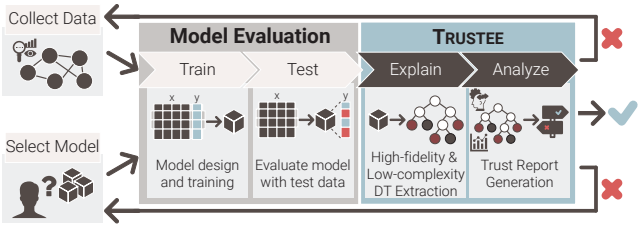

Figure 1: TRUSTEE overview.

At the same time, this basic tension has also driven recent efforts to "crack open" the black-box learning models, explaining why and how they make their decisions (*e.g.,* "interpretable ML" [52], "explainable AI (XAI)" [60], and "trustworthy AI" [12]). However, to ensure that these efforts are of practical use in particular application domains of AI/ML such as network security is challenging and requires further qualifying notions such as (model) interpretability or trust (in a model) [41] and also demands solving a number of fundamental research problems in these new areas of AI/ML.

In this paper, we first provide such a qualification that is motivated by the needs of the field of network security as application domain of AI/ML and equates *"an end user having trust in an AI/ML model"* with *"an end user being comfortable with relinquishing control to the model"* [41]. Given this specific definition of what it means for an AI/ML model to engender trust, we next address a number of fundamental research challenges related to the problem of quantitatively deciding when an end user is comfortable with relinquishing control to a given AI/ML model. To this end, a particular focus of this paper is on determining whether or not a given AI/ML model suffers from the *problem of underspecification* [17].

Here, the problem of underspecification in modern AI/ML refers to determining whether the success of a trained model (*e.g.,* high accuracy) is indeed due to its innate ability to encode some essential structure of the underlying system or data or is simply the result of some inductive biases that the trained model happens to encode. In practice, inductive biases typically manifest themselves in instances of shortcut learning strategies [29], signs of spurious correlations [3], or an inherent inability for out-of-distribution (o.o.d.) generalizations (*i.e.,* test data distribution is different from training data distribution). The implication of such inductive biases is that their presence in trained AI/ML models prevents these models from being credible or trustworthy; that is, generalize as expected in deployment scenarios. Thus, for establishing the specific type of trust in an ML model considered in this paper, it is critical to be able to identify these inductive biases, and this paper takes a first step towards achieving this ambitious goal.

To detect underspecification problems in learning models for network security problems, we develop TRUSTEE (TRUSt-oriented decision TreE Extraction). This framework provides a means for carefully inspecting black-box learning models for the presence of inductive biases. Figure 1 shows how TRUSTEE augments the traditional ML pipeline to examine the trustworthiness of a given ML model. Specifically developed with the application domain of network security in mind, TRUSTEE takes a given black-box model and the dataset that has been used to train that model as input and outputs a "white-box" model in the form of a high-quality decision tree (DT) explanation.

Importantly, in synthesizing this DT, TRUSTEE's focus is first and foremost on ensuring its practical use which, in turn, requires leveraging domain-specific observations to strike a balance between *model fidelity* (*i.e.,* accuracy of the DT with respect to the black-box model), *model complexity*, and *model stability*. Here, complexity refers to both the size of the DT and to aspects of the tree's branches. In particular, when viewing the tree's branches as decision rules, we are concerned with their explicitness and intelligibility; that is, we require these rules to be readily recognizable by domain experts and be largely in agreement with the experts' domain knowledge. Model stability, on the other hand, is concerned with the correctness, coverage and stability of the decision rules; that is, we require them to correctly describe how the given black-box model makes a significant number of its decisions and also want them to be largely insensitive to the particular data samples that TRUSTEE used in the process of selecting its final DT explanation. We achieve this insensitivity or stability by implementing a heuristic method that selects from among a number of different candidate DTs the one that has the highest mean agreement. Here, the agreement between two different DTs is a measure of how often the two DTs will make the same decision for the same input data [31, 61]. In practical terms, implementing this heuristic reduces the likelihood that TRUSTEE outputs a misleading DT explanation.

TRUSTEE also outputs a trust report associated with the DT explanation, which operators can consult to determine whether there is evidence that the given black-box model suffers from the problem of underspecification. If such evidence is found, the information provided in the trust report can be used to identify components of the traditional ML pipeline (*e.g.,* training data, model selection) that need to be modified in an effort to improve upon an ML model that TRUSTEE has found to be untrustworthy.

While our work contributes to the rapidly growing ML literature on model explainability/interpretability and is inspired by ongoing developments in this area, our efforts and objectives differ from existing approaches in a number of significant ways. For one, given the inherent complexity of learning problems for networking, existing approaches of replacing black-box models with "white-box" models that are inherently explainable in the first place (*e.g.,* decisions trees) are in general impractical. Moreover, *local interpretability* methods [32, 49, 54] are not suitable for examining the various instances of the underspecification problem. At the same time, although our effort is motivated by prior studies that focus on *global interpretability*, [6, 7, 38]) these works are either only applicable to a specific class of learning models (*e.g.,* reinforcement learning) or suffer from poor fidelity.

Through various case studies, we illustrate in Section 7 how operators can use TRUSTEE's DTs and associated trust reports to detect the presence of inductive biases. More specifically, we use published ML models that are reproducible (*i.e.,* code base and datasets are publicly available) to show how network operators can use the information provided by TRUSTEE to detect instances of shortcut learning strategies, obtain evidence of overfitting and/or whether the model relies on spurious correlations to make its decisions, or determine the model's inability to generalize for out-of-distribution data.

2 BACKGROUND AND RELATED WORK

The application domain of AI/ML considered in this paper is the area of network security. In this section, we first discuss the unique challenges that this area poses for utilizing the latest advances in AI/ML. In particular, we focus on important recent AI/ML concepts such as "interpretable ML" and "explainable AI" and discuss their relevance for our work.

2.1 Challenges in ML for Network Security

Beyond the already-mentioned trust issue, there are a number of other reasons why the area of network security is a particularly challenging application domain for AI/ML. Networking-related datasets in general and cybersecurity-specific datasets in particular typically contain information about what is being communicated over a network (*e.g.,* packet-level traffic traces) or provide insight into how networks enable such information exchanges. As such, the datasets often raise serious end user-specific privacy concerns or reveal provider-specific details that many companies consider to be proprietary in nature and are therefore unwilling to share. The result is a general paucity of publicly available datasets. Moreover, the datasets that are publicly available generally lack the complexity of real-world settings, either because they have been synthetically generated, have been obtained from small-scale testbed environments, or have been anonymized to the point where their general utility has been severely curtailed.

The scarcity of carefully labeled data poses an even bigger problem. Networking or cybersecurity datasets do not come in the form of images that humans can recognize but typically consist of semantically rich content, and unpacking that content and properly labeling it often requires substantial domain knowledge (*e.g.,* network architecture, protocols, standards). This need for domain knowledge rules out labeling approaches that have been used successfully in other domains and include crowdsourcing (*e.g.,* for labeling images that are part of open-source databases such as ImageNet [18]) or outsourcing (*e.g.,* for labeling datasets that have been curated and open-sourced by commercial self-driving car companies for the benefit of researchers [14, 15]).

2.2 Interpretable ML and Explainable AI

As the scientific community continues to develop sophisticated AI/ML-based tools for high stakes decision-making throughout society, there has been a growing awareness about their actual or potential misuses and negative implications. As a result, calls for starting to study "trustworthy AI", "responsible AI", "ethical AI" and related topics have intensified in recent years and have identified model interpretability/explainability as a critically important but also highly elusive concept for facilitating these studies [41].

Interpretable ML: Ex-ante Interpretability. The application of modern AI/ML has resulted in a myriad of different learning models that are "black-box" in nature; that is, provide no insight in or understanding about why the black-box model makes certain decisions (and not some other decisions) or what decision-making process gives rise to these decisions. This development has resulted in a recent explosion of work on "Explainable AI," where a second

(post-hoc) model is created to explain the originally obtained black-box model [60]. This pursuit of explainable AI has been criticized in the recent AI/ML literature and called "problematic" (see, for example [52]), mainly because such post-hoc explanations are often not reliable and can be misleading [30, 38]. An alternative approach that has been advocated in [52] argues for using learning models such as linear models or DTs that are inherently (*i.e.,* ex-ante) interpretable.

Unfortunately, because of the rich semantic content of the data, in the network security domain, uncovering the types of patterns in the data that matter has become increasingly the responsibility of trained "black-box" models rather than painstakingly-designed inherently explainable models. However, instead of considering this development as being "problematic", we view it as an unique opportunity to ultimately achieve the vision of interpretable ML – ensuring that AI/ML models used for high stakes decision-making are fully comprehensible by their end users and interested third parties.

Explainable AI: Post-hoc Interpretability. A commonly-made argument in favor of using black-box models such as deep neural networks or random forests is that they typically achieve higher accuracy compared to their interpretable counterparts (*e.g.,* DTs) and are therefore often more desirable when used in practice. Although this argument is not universally shared (*e.g.,* see [52]), it nevertheless has been a driving force behind the recent efforts on the topic of "explainable AI." Also referred to XAI [60], explainable AI describes efforts where the development of a trained black-box model is followed up with additional activities that are intended to help "explain" the originally obtained black-box model. These efforts can be divided into two disjoint categories, namely *local explainability* and *global explainability*.

Methods for providing local explanations aim at illuminating how a black-box model makes individual decisions (or decisions in a local region near a particular data point) and include well-known techniques such as LIME [49], SHAP [42], and LEMNA [32]. Since these methods limit their attention to only a subset of individual decisions, they are prone to providing misleading explanations [41, 46, 65], depending on the subset of samples analyzed. Related methods such as Partial Dependence Plots (PDP) [27] and Accumulated Local Effect (ALE) plots [1] suffer from similar shortcomings. As such, these methods are of limited use when we seek explanations that we can trust in the sense that they accurately describe how a given black-box model makes decisions holistically.

It is methods that provide global explanations that aim at describing how a given black-box model makes its decisions "as a whole" and not one data sample at a time. Extracted from the black-box model in a second step (*i.e.,* post-hoc), such explanations typically take the form of an inherently interpretable model such as a rule set or a DT [6, 39] and become the main vehicles for studying the decision-making process of the original black-box model and examining its properties. However, existing approaches for such post-hoc extractions of global explanations are known to produce at times too low of a fidelity to be useful in practice [6], target only a very specific set of black-box models [7], be difficult to reproduce [38, 39], and be possibly unreliable to the point of being misleading [30, 38]. To achieve the level of explainability required in high-stakes application domains such as network security, we seek to generate high-fidelity global explanations that are capable of accurately and faithfully describing a majority of the decisions made by any given black-box model.

3 TRUSTEE OVERVIEW

Our focus in this paper is on post-hoc global model interpretability for the application domain of network security problems. Although the idea of using DTs for investigating global model interpretability for a given black-box model is not novel, the set of requirements that we impose on the DT explanations we desire is non-standard, makes this a challenging problem, and motivated us to develop TRUSTEE.

For one, we require that our new DT extraction method be model-agnostic; that is, applicable to any given black-box model. Second, we also demand that the method produces high-fidelity DT explanations; that is, DTs whose expected predictive performance is similar to that of the black-box model. To quantify the fidelity of DTs, we rely on well-known metrics; for example, while for classification problems, we measure fidelity using the F1-score between classifications from the black-box model and the DT, for regression problems, fidelity is measured in terms of the R-squared value between the predictions from the black-box and the DT. The third requirement we impose is that the extraction method also results in low-complexity DT explanations such that selected parts of the DTs are intelligible and comprehensible (*i.e.,* easy to understand by domain experts) and accurately describe how the black-box model makes most of its decisions. The fourth and last requirement concerns a stability property that we want our new DT extraction method to have. In particular, to reduce the chances that this output provides a misleading DT explanation, we require that most of the final DT's decisions should be insensitive to the minute details of how this final DT explanation has been determined.

For a DT that TRUSTEE extracted from a given black-box model and that satisfies this set of requirements, our next goal is to summarize the pertinent aspects of this synthesized tree in a trust report. This trust report is intended to help end users determine whether the given black-box model suffers from the problem of underspecification and cannot be trusted. To achieve this goal, we look for ways to exploit the extracted DT explanation for the purpose of enabling the end users to investigate the black-box model for likely indications of the presence of inductive biases. In particular, in this paper we consider the following three instances of inductive biases: *(i)* instances of shortcut learning, *(ii)* signs of spurious correlations, and *(iii)* problems with out-of-distribution samples.

Note that the presence of any of these inductive biases proves that the given black-box model suffers from the underspecification problem and cannot be trusted. At the same time, the absence of these instances does not mean that the black-box model can be trusted. In fact, while proving for an arbitrary black-box model that the model does not suffer from the underspecification problem is hard and remains an unsolved problem, showing that the model does suffer from the underspecification problem only requires demonstrating the presence of a single instance of an inductive bias, and our design of TRUSTEE is an initial effort that simplifies demonstrating that certain biases are present in a given model. After describing TRUSTEE's design in detail in Section 4, we illustrate the end-to-end application of TRUSTEE, including the use of the extracted DT explanation and resulting trust report with a number of illustrative examples in Section 7.

4 EXTRACTING DECISION TREES

The first step to realize the agenda detailed in Section 3 consists of generating high-fidelity and inherently interpretable (*i.e.,* "white-box") counterparts for any given black-box model, regardless of the learning method used by the black-box. To this end, we first discuss

existing approaches to this problem and their limitations. We then present TRUSTEE, an original and practical framework that end users can apply to extract high-fidelity DT explanations from an arbitrary black-box learning model.

4.1 Existing approaches

Global white-box explanations extracted from a black-box model can often describe in detail the reasoning behind the model's behavior, provided they achieve a good enough fidelity. Earlier works [6, 7, 16, 44] have proposed different approaches to extracting DT explanations from black-box models, but these DTs typically do not satisfy all the above-listed requirements and are therefore ill-suited for end users who want to gauge their level of trust in a given black-box model. We list a number of relevant prior efforts and their pertinent features in Table 1. Note that some of these existing methods [7, 44] are not model-agnostic but have been designed for specific learning paradigms and models, such as Reinforcement Learning. As such, they typically rely on assumptions that are specific to the learning paradigm or model that their designs focus on. On the other hand, prior efforts that do propose model-agnostic approaches [6, 16] tend to produce DT explanations that don't satisfy the fidelity requirement that we demand for realizing our objective (see Appendix B for empirical evidence).

Table 1: Existing approaches to extract decision trees.

Method	Optimization Objective	Stopping Criterion	Model Agnostic	High Fidelity	Domain-specific Pruning
Trepan [16]	Fidelity	Max Nodes	✓	-	-
dtextract [6]	Accuracy	Max Nodes	✓	-	-
VIPER [7]	RL Reward	Max Iterations	-	-	-
Metis [44]	RL Reward	Max Iterations	-	-	-
TRUSTEE	Fidelity	Max Iterations	✓	✓	✓

Another important aspect of many of these existing efforts is the stopping criterion they use to obtain their extracted DT explanations. For example, prior efforts such as [6, 16] require specifying the maximum size (*i.e.,* number of nodes) that the extracted DT can have and use this input parameter as stopping criterion. Such approaches are convenient for obtaining explanations that are guaranteed to be of a certain size, but this convenience typically comes at the cost of low fidelity, implying that important decision-making rules may be missing from the resulting DT. Other methods such as [7] and [44] extract DT explanations in an iterative fashion, require specifying the maximum number of iterations, and use this user-specified input parameter as stopping criterion. Even though these methods do not explicitly optimize for fidelity, they typically produce high-fidelity explanations, but at the cost of high complexity (*i.e.,* the large size of the resulting explanations makes interpreting cumbersome if not impractical). To overcome this problem, the authors of [44] rely on a commonly-used technique called Cost-Complexity Pruning (CCP) [27]. Similar to other pruning methods [25], CCP succeeds in striking a balance between the overall fidelity of the extracted DTs and their size. However, from an interpretability perspective, CCP tends to be oblivious to what roles what decision-making rules play as part of the resulting DT explanations. Because of this observed trade-off between model complexity and model interpretability, these methods are ill-suited for our purpose where we strive to shed light on the decision-making rules that are key to interpreting black-box models that arise in the context of solving network security-related problems.

4.2 Model-Agnostic Decision Tree Extraction

Given the absence of readily available model-agnostic methods for extracting high-fidelity, low-complexity, and stable DTs from black-box ML models, we present in the following TRUSTEE. Algorithm 1 describes the steps that TRUSTEE takes to achieve its objective.

At a high level, these steps are executed as part of an outer loop (lines 4-16) that is executed a total of S times. Each iteration of this outer loop involves an inner loop (Lines 5-12) that is performed N times. Here, this inner loop is designed to generate different high-fidelity DT explanations, one per iteration. It does so by applying a teacher-student dynamic derived from imitation learning [34] that uses π^* as an oracle in conjunction with a carefully curated dataset $\mathcal{D}'$ to guide the training of a surrogate "white-box" model in the form of a DT that imitates the black-box's decisions. In contrast, the purpose of the outer loop is *(i)* to select from among the N high-fidelity DTs that have been generated in the process of executing the inner loop the DT with the highest fidelity, *(ii)* to transform this resulting DT into a high-fidelity and low-complexity DT by means of a post-processing step that consists of applying a purposefully-developed pruning method (see Section 5 for details), and *(iii)* to consider all S high-fidelity and low-complexity DTs that have been generated in the process of executing the outer loop and output the one that is the most stable in the sense of having the highest mean agreement among these S DTs.

Algorithm 1 takes as input a given black-box model π^* that we desire to explain and the original dataset $\mathcal{D}_0$ that was used to train π^*. Other parameters that the algorithm requires as input are the number of iterations S for the outer loop, the number of iterations N for the inner loop, the number of samples M to select from $\mathcal{D}_0$ to use when training the surrogate DTs as part of each iteration of the inner loop, and a parameter k that is required by the tree pruning method used in Line 14 and is discussed in more detail in Section 5 below. The algorithm starts by initializing the training dataset $\mathcal{D}$ (Line 2) using the given black-box π^* to predict the expected outcomes from the given input data $\mathcal{D}_0$. It then initializes a set of DTs (Line 3) from which, at the end (Line 17), the most stable DT explanation will be selected and returned as output by TRUSTEE (Line 18).

To execute the inner loop as part of an iteration of the outer loop, the steps that the algorithm performs during the j-th ($1 \leq j \leq N$) iteration of the inner loop consist of *(i)* selecting M training samples uniformly at random from the optimal prediction dataset $\mathcal{D}$ (Line 6) to initialize a training dataset $\mathcal{D}'$; *(ii)* splitting the dataset $\mathcal{D}'$ for training and testing (Line 7); *(iii)* training a DT student $\hat{\pi}_i$ on $\mathcal{D}'_{train}$ (Line 8) by using the well-known CART method [11]; *(iv)* testing the DT explanation using $\mathcal{D}'_{test}$, collecting the samples that the DT classifier wrongly classifies into the set $\mathcal{D}'_e$ (Line 9), and querying the black-box model for the expected results for $\mathcal{D}'_e$ (Line 7) to produce a correction dataset $\mathcal{D}_j$ (Line 10); and *(v)* augmenting the optimal dataset $\mathcal{D}'$ with this correction dataset (Line 11) to reinforce the correct decisions during the subsequent iterations of the inner loop.

The steps that the algorithm executes during the i-th ($1 \leq i \leq S$) iteration of the outer loop are *(i)* perform N iterations of the inner loop (Lines 6-11), *(ii)* select from among the N generated different student models the one DT explanation with the highest fidelity (Line 13), and *(iii)* apply a special pruning method to this highest-fidelity DT to obtain a high-fidelity and low-complexity DT explanation candidate. Finally, after S iterations of this outer loop, the algorithm selects from among the S obtained different high-fidelity and low-complexity DT explanation candidates the one which has the highest

mean agreement (*i.e.,* is the most stable) and returns this "best of the best" DT as final output of TRUSTEE.

In the following, we provide a more detailed description of the main design choices we made for TRUSTEE and further evaluate some of these design choices as part of an ablation study in Section 8.

Algorithm 1 Model agnostic decision tree explanation extraction.

1: **procedure** TRUSTEE(
   π^*: Black-box model,
   $\mathcal{D}_0$: Initial training dataset,
   M: Number of samples to train the decision tree,
   N: Number of iterations of inner loop,
   S: Number of iterations of outer loop,
   k: Parameter for Top-k Pruning),
2: Initialize dataset using black-box $\mathcal{D} \leftarrow \pi^*(\forall x \in \mathcal{D}_0)$
3: Initialize stabilization set of DTs $\mathcal{R} \leftarrow \emptyset$
4: **for** $i \leftarrow 1 \ldots S$ **do**
5: **for** $j \leftarrow 1 \ldots N$ **do**
6: Sample M training cases uniformly from $\mathcal{D}$
$\mathcal{D}' \leftarrow \{(x, y) \overset{\text{i.i.d.}}{\sim} U(\mathcal{D})\}$
7: Split sampled dataset for training and testing
$\mathcal{D}'_{train}, \mathcal{D}'_{test} \leftarrow$ TRAINTESTSPLIT($\mathcal{D}'$)
8: Train DT
$\hat{\pi}_j \leftarrow$ TRAINDECISIONTREE($\mathcal{D}'_{train}$)
9: Test and get samples DT misclassifies
$\mathcal{D}'_e \leftarrow \{\forall (x, y) \in \mathcal{D}'_{test} \mid \hat{\pi}_j(x) \neq \pi^*(x)\}$
10: Get correct outcome from black-box
$\mathcal{D}_j \leftarrow \pi^*(\forall x \in \mathcal{D}'_e)$
11: Augment dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_j$
12: **end for**
13: Select tree with highest fidelity
$\hat{\pi}_{max} \leftarrow \hat{\pi} \in \{\hat{\pi}_1, ..., \hat{\pi}_N\}$
14: Prune selected tree $\hat{\pi}_i \leftarrow$ TOPKPRUNE($\hat{\pi}_{max}, k$)
15: Add tree to the stabilization set $\mathcal{R} \leftarrow \mathcal{R} \cup \hat{\pi}_i$
16: **end for**
17: Select tree with highest mean agreement with others
$\hat{\pi}_{agree} \leftarrow \hat{\pi} \in \mathcal{R}$
18: **return** $\hat{\pi}_{agree}$
19: **end procedure**

Multiple iterations and uniform sub-sampling. The CART algorithm that is traditionally used to train DT models relies on a greedy approach for finding the best splits in the given training dataset. This greedy approach ensures that for a given training dataset, the resulting DT will be largely insensitive to the order in which the input samples are processed. At the same time, this greedy approach is prone to produce over-fitted DTs [10]. While using this approach without further constraints (*e.g.,* stopping criterion) to train a DT results in perfect fidelity, being over-fitted makes the resulting DT ill-suited for providing an intelligible explanation for how the given black-box model makes its decisions. Instead, the resulting DT explanation is largely an artifact of the method used to generate the explanation. To overcome this problem, TRUSTEE implements an iterative approach to train multiple student models on the expert model predictions. This iterative approach is implemented as the inner loop in Algorithm 1, where at each iteration, we select a fraction M of the input data by uniform sub-sampling from the original training dataset (Lines 5-11). This approach differs from existing efforts [7] in that by requiring the uniform sub-sampling step at each iteration, we ensure that each DT explanation will have only a limited view of the entire data, akin to a k-fold cross validation [53]. Incorporating this sub-sampling step allows us to stress-test how different features and/or feature values contribute to the decision-making of the black-box model and then select the ones that best fit our overall objective. In practical terms, uniform sub-sampling from the original training dataset assumes each sample has the same probability of being selected (*i.e.,* balanced dataset). While it is well known that using imbalanced datasets to train ML models leads to biases towards the majority classes, the existing ML literature provides several approaches that resolve this problem through proper pre-processing of the original training data [36].

Dataset augmentation. An important design choice for TRUSTEE involves a dataset augmentation step (Line 9), where in each iteration of the inner loop, the algorithm uses the optimal predictions from the black-box model on the sampled dataset D' to augment the original training dataset D. The purpose of this step is to over-correct for data samples for which the student DT model makes wrong decisions. Leveraging results from the existing literature on imitation learning [7, 51], performing this step can not only increase the overall accuracy of the trained student model but also reduce the overall number of leaf nodes in the resulting tree.

Fidelity as objective function. When selecting from among the different student models that TRUSTEE extracts from a given black-box model, it uses model fidelity as objective function and picks the student model with the highest fidelity (Line 11). This design choice implies that while the final DT explanation produced by TRUSTEE is not necessarily the most *accurate* DT for the given black-box model, it is the DT that is the most *faithful* one in terms of explaining how the black-box model makes its decisions. Intuitively, it is by insisting on this high-fidelity aspect of the DTs that TRUSTEE considers that we are able to post-process the resulting DT explanation in ways that will help end users with varying degrees of domain knowledge to gauge their trust in the given black-box model. We provide evidence in support of this intuition in Section 5 where we describe the type of post-processing that we perform as part of TRUSTEE so the final DT explanation it outputs can serve as an inherently practical means for faithfully explaining most of the given black-box model's decisions.

Model stability. Since the inner loop of Algorithm 1 (Lines 6-11) uses a different random subset of the entire dataset each time it trains a DT explanation, it is possible for TRUSTEE to output a misleading explanation because of the particular subset of data that was used to train that final DT explanation. To minimize the chances for such scenarios to occur, we add an outer loop in Algorithm 1 (Lines 4-16). This addition results in the extraction of S different high-quality DT explanations from the given black-box model and allows us to measure the agreement among these S different DT explanations. The agreement of DTs is a well-known measure of how often a pair of DTs will make the same decisions for the same input data and is a metric that has been used in previous studies that concern assessing the stability of different DTs [31, 61]. Here, to select the final DT explanation that is returned as output of TRUSTEE (Line 18), Line 17 in Algorithm 1 computes the pair-wise agreement among all S DT explanations and selects the one with highest mean agreement. While implementing this outer loop prevents TRUSTEE from generating obviously misleading explanations and gives domain experts confidence that they can trust the explanations produced by TRUSTEE's output, rigorously proving that a "white-box" model extracted from a given black-box model does not provide misleading explanations is an active area of current research [38].

5 PROCESSING DECISION TREES

When using TRUSTEE to synthesize a high-fidelity DT explanation for a given black-box model, realizing the agenda outlined in Section 3 requires performing an additional step in Algorithm 1 (Line 14) each time N iterations of the inner loop have completed and the algorithm has selected the highest-fidelity DT from among the resulting N different high-fidelity DT candidates (Line 13). The purpose of this step is to transform this selected highest-fidelity DT into a DT explanation for the given black-box model that is inherently practical in the sense of having low complexity and at the same time high fidelity. Here, when referring to the complexity of a DT explanation, we mean small trees but also, and more importantly, trees whose main branches (*i.e.,* decision rules ranked by number of input samples they classify) explicitly, intelligibly, and accurately describe how the black-box model makes most of its decision. Effectively, when generating this low-complexity and high-fidelity DT explanation as a result of this post-processing step, we tolerate some loss of fidelity in return for achieving low complexity. In the following, we examine different aspects of this fidelity-complexity trade-off and introduce a simple tree pruning method that we call *Top-k Pruning* and that comprises the required post-processing step. We design this method for the explicit purpose of ensuring that any final DT explanation that TRUSTEE outputs can be readily processed and understood by domain experts.

5.1 Decision Tree Pruning: Trade-offs

One of the main disadvantages of CART models is that CART's greedy algorithm is known to be prone to overfitting, often producing high-fidelity DTs that can have thousands of nodes [25]. Clearly, such large trees are detrimental to our ultimate goal; that is, presenting end users with inherently practical explanations that they can readily inspect and understand with their available domain knowledge. In designing TRUSTEE, we similarly focused on first obtaining largely unconstrained DT explanations with the best possible fidelity (Line 13). However, our reasoning for doing so is that we explicitly require that any DT explanation that TRUSTEE outputs will have undergone a post-processing phase for the purpose of making this final DT explanation intelligible and comprehensible for end users. Our intuition behind obtaining a high-fidelity DT explanation first and addressing its complexity later is that manipulating a high-fidelity explanation with an eye towards reducing its complexity is more likely to result in explanations that, while experiencing some decrease in their fidelity, still will have higher fidelity than their counterparts that had lower fidelity to start with.

A commonly-used approach to transforming large DTs into trees of smaller sizes is pruning, and the existing literature describes several pruning methods for DTs [25, 27], many of which are highly effective in obtaining DTs that have small complexity, at least as far as the size of the trees are concerned. Among the most widely-used approaches to pruning are *(i)* pre-pruning which limits either the total number of nodes or the overall depth of the tree and *(ii)* post-pruning, such as Cost-Complexity Pruning (CCP) [27]. On the one hand, by explicitly constraining either the number of tree nodes or the tree's depth, the pre-pruning approaches allow for direct control over the size of the resulting DT. However, this control over tree size typically comes at the cost of reduced fidelity, mainly because the obtained small trees run the risk of missing important decision branches as they prevent the consideration of any further decision branches once the stopping criterion is reached. On the other hand, using post-pruning such as CCP often results in a better trade-off between fidelity and size. However, this better trade-off comes at the cost of reduced interpretability, mainly because CCP relies on a parameter that is indirectly responsible for how many nodes of a tree get pruned. Lack of a more direct control makes it difficult to decide which tree branches to include in the pruned tree and which to exclude.

5.2 Top-k Pruning Method

Each branch in any of the highest-fidelity DTs that are selected in the process of executing an iteration of Algorithm 1's outer loop (*i.e.,* Line 13) represents a decision "rule"; that is, a combination of individual decisions on features that results in labeling the input data as belonging to a specific class (*e.g.,* malware vs. benign). Since each of these "rules" accounts for a certain percentage of all samples in the input dataset, the different rules also contribute differently to the overall model fidelity, and as the complexity of the DT grows (*i.e.,* larger number of branches), so does the DT's fidelity.

The idea behind our Top-k Pruning method is that to detect signs of the presence of inductive biases in a given black-box model, it often suffices to carefully scrutinize only the top-k branches of an extracted high-fidelity DT, ranked by the number of input samples the branch classifies, especially in cases where the branches intelligibly describe how the black-box makes most of its decisions. In particular, we argue that the "tail" end of the branches of an extracted high-fidelity DT (*i.e.,* branches that are not in the top-k for some large value of k) often reflects specific decisions of the black-box model that are overfitted to the training dataset and can, for all practical purposes, be ignored when trying to explain the most important decisions of the black-box model. However, since the trade-offs between model fidelity and model complexity are typically model dependent,Top-k Pruning requires a parameter k, and specifying a value for k gives end users complete control over how many branches they want to consider in their attempt to understand the trade-offs for a given model. Also note that even though selecting smaller k values can possibly result in poor fidelity, it does not mean that we cannot draw potentially important conclusions from the resulting explanation. For instance, one specific branch can sometimes cover most branches of a particular class, resulting in an apparent poor overall fidelity but still indicating a potential underspecification issue related to that specific class. In short, the user-specified parameter k determines the complexity of the final DT explanation that TRUSTEE presents to the end user. If, at any point, a user wants to inspect more branches of the tree, they can simply choose a larger k and rerun our algorithm. Note, however, that due to its probabilistic nature, re-running our algorithm will typically result in applying the Top-k Pruning method to a high-fidelity DT explanation that differs from the original one. We leave a careful investigation of this aspect of TRUSTEE and its deeper implications for detecting instances of inductive biases in a given black-box model for future work.

5.3 Generating Trust Reports

We use the DT explanation that TRUSTEE outputs as basis for populating a trust report that simplifies the task of end users of a given black-box model to gauge their trust in that model. In the following, we provide details on how we build this trust report so it helps end users spot signs that point to possible instances of inductive biases in the given black-box model. If upon further scrutiny of these signs, the presence of such an inductive bias is confirmed, it would be proof for the end users that they cannot trust the given model. To this end, we leverage the fact that any DT explanation that TRUSTEE outputs has been pruned with the help of our Top-k Pruning method

and is therefore typically a small tree comprised of k branches. As part of the trust report, we present the details of the generated small DT to the end users so they can examine these details with an eye towards three common ways an underspecified ML model can be recognized More precisely, we intend the trust report to be the main source of information that end users can consult when checking for inductive biases that manifest themselves as instances of shortcut learning strategies, through the presence of spurious correlations, or in an inability to generalize for realistic out-of-distribution data. Importantly, by itself, the information contained in the trust report is in general insufficient to diagnose underspecification issues; instead, it points to potentially attention-worthy aspects of the model or the data that require further attention. As such, detecting and diagnosing underspecification issues is not a task that is currently automated but requires domain knowledge and great familiarity with the learning problem at hand. Consequently, the effort demands a (human) domain expert to actively inspect and check if the trust report-provided information points to possible problems in the data or in the model, or indicates that there are no problems with either the model or the data. In the following, we briefly describe how the generated trust report helps with checking for each of these three inductive biases.

Shortcut learning. Presenting a visual depiction of the small DT explanation that forms the output of TRUSTEE and annotating it with pertinent information (*e.g.,* features used, splitting conditions or clauses present at the different nodes of the tree, number of input samples associated with each branch segment) allows for quick and intelligible perusing and inspection of the tree. In particular, observing that less than a handful of input features are required to accurately classify most of the input data (or a specific class of input samples) is often a strong indication of a shortcut that the black-box model learned and that can in general quickly be confirmed with readily available domain knowledge. Note however that a small number of features in the output of TRUSTEE may also indicate that the learning problem for which the black-box was designed for in the first place is in fact simple and may not require any ML at all.

Spurious correlations. A more involved investigation of the information provided in the trust report concerns studying the impact of removing the identified most important feature(s) from the provided dataset. Upon removing such feature(s), we can then retrain the black-box model using this altered dataset, proceed to use TRUSTEE to extract a new DT explanation, and repeat this process a number of different times. In general, the impact of removing important features from the training dataset is that the accuracy of the black-box model decreases. However, especially in situations where the data include a large number of features, it is often the case that the back-box model is able to find alternative features so that removing the most important feature(s) leaves the overall model accuracy essentially unchanged. We take this as a strong indication of the presence of spurious correlations in the data that can subsequently be easily confirmed via an analysis of the original feature set.

Out-of-distribution samples. The annotated version of the output of TRUSTEE that is shown as part of the trust report can also be utilized to consider the individual features in each of the tree branches, plot the distribution of the values that each feature can take in the provided dataset, and include the different distribution plots in the trust report. Inspecting the resulting distributions affords end users an opportunity to reason whether or not the observed distributions of feature values are consistent with those encountered in data collected from actual production settings. Such an inspection is especially informative when the provided datasets consist of network traffic measurements, where feature value distributions are typically dictated by the dominant protocols in use, where artifacts can often be easily identified (*e.g.,* due to simple testbed experimentations), and where generating meaningful out-of-distribution samples is in general feasible because the expected behavior across the full TCP/IP protocol stack is either known or well documented.

6 USING TRUSTEE IN PRACTICE

The following step-by-step instructions are intended to help end users who want to use TRUSTEE and associated trust reports to check if a given trained ML model is credible or not by inspecting it for possible underspecification issues.

Step 1 (Getting started): Select the ML model that needs to be analyzed and the dataset of the input samples that is used to examine the model's decisions and decision-making process. The only requirement for the selected ML model is that it provides a `predict` interface that TRUSTEE can use to query the model for its prediction for a given input sample. As for the input dataset, TRUSTEE also accepts datasets that differ from the dataset used to train the ML model, but we recommend using the training dataset for a basic analysis of the selected ML model.

Step 2 (Selecting hyperparameters): TRUSTEE requires selecting values for four hyperparameters: S and N (number of iterations of the outer loop and inner loop in Algorithm 1, respectively), M (sampling rate), and k (number of branches to keep as part of our Top-k Pruning method). Although highly model- and data-dependent, we found that in the context of the different use cases we analyzed, choosing $S = 10$, $N = 50$, $M = 30\%$ of the input dataset, and $k = 10$ is a good starting point and allows for subsequent modifications should the need arise. In general, we recommend selecting suitable values for M, S,and N by qualitatively comparing the learning curves [53, Chapter 18.3.3] and checking the DT fidelity between training and test data for different hyperparameter values, but more quantitative methods such as grid search [8] or Bayesian Optimization [58] could be used as well. However, the number of samples M is highly dependant on the size and type of the available data; setting a sampling rate too high increases the risk of over-fitting, setting the sampling rate too low increases the risk of under-fitting and is likely to require a higher number S of iterations of the outer loop of Algorithm 1. In turn, selecting the parameter k (*i.e.,* number of top-k branches) depends first and foremost on the amount of information a domain expert is willing to analyze when presented with TRUSTEE's output.

Step 3 (Detecting underspecification issues): Identifying the presence and/or nature of underspecification issues in a model currently requires manual inspection by a domain expert. The main vehicle for performing this manual task is the DT explanation produced by TRUSTEE in conjunction with the information provided in the corresponding trust report. Relying on basic trust report-provided information can often point to potential biases, but it typically invites further scrutiny at the level of individual decision rules (*i.e.,* tree branches) where, for example, certain deficiencies in the training dataset (*e.g.,* missing samples of real-world patterns or behavior) can be identified.

Step 4 (Validating DT explanations): Validating a DT explanation that forms the output of TRUSTEE typically requires tinkering with the ML model itself, with the feature engineering as part of the model's design, or with the provided dataset. Unfortunately, a general inability to easily collect new or different data severely limits the validation efforts that require modified data. In such cases, we found that tampering with the original data (*e.g.,* removing certain

features or artificially modifying packet headers in a trace) can be a viable option but has to be done with care to ensure that the tampered dataset consists of realistic input samples that the ML model ought to be able to handle.

7 RESULTS

In this section, we illustrate with different use cases how TRUSTEE can be used in practice. Each use case concerns a recently published black-box model that has been developed for a particular network security-related problem and is accompanied by open-sourced artifacts (*e.g.,* code base, dataset) that are required for reproducing the reported findings and assessing whether the ML model is credible.

7.1 Summary

Table 2 summarizes the use cases we analyze. The first use case (§7.2) illustrates how an apparently high-performant neural network learns simple shortcuts to distinguish between two types of traffic (VPN vs. Non-VPN). It highlights the importance of having an in-depth understanding of the data used to train a model. The second use case (§7.3) analyzes a black-box model (*i.e.,* random forest) trained using the popular synthetic dataset CIC-IDS-2017 [55] and shows that the developed model is vulnerable to o.o.d. samples. This use case cautions against an over-reliance on synthetic datasets that often include measurement artifacts that commonly-considered black-box models exploit to achieve high accuracy. The third use case (§7.4) analyzes a recent approach that advocates using bit-level feature representations of the input data instead of carefully engineered and semantically meaningful features [33]. This use case warns against the indiscriminate use of the high-dimensional feature spaces that result from such representations because they allow black-box models to identify and exploit spurious correlations between features to achieve high accuracy. The fourth use case (§7.5) concerns the application of a complex ensemble of neural networks [45] to perform traffic anomaly detection (*e.g.,* Mirai attack). By showing that this model is also vulnerable to o.o.d. samples, we corroborate previously-reported criticism of this model [4] and support it with further evidence. The remaining use cases listed in Table 2 are analyzed in Appendix D. All datasets, models, and results for all seven use cases are available at [35].

7.2 Detecting VPN vs. non-VPN Traffic

Problem setup. We consider the paper [62], which presents an AI/ML-based framework for encrypted traffic classification that integrates feature design, feature extraction, and feature selection. It uses one-dimensional convolutional neural networks (1D-CNN) to automatically learn the relationships between raw packets and the output labels. For classifying VPN vs. Non-VPN traffic, the authors train a 1D-CNN learning model with the PCAPs of the ISCX VPN-nonVPN dataset [21], treating the packets of each session as a 2D image of size 28x28. As a result, the proposed model views the input traffic samples as discrete byte streams of fixed length (*i.e.,* 784 bytes) and treats each byte as a "feature". [62] reports outstanding performance (*i.e.,* 100% (99.9%) precision and 99.9% (100%) recall for Non-VPN (VPN) traffic). All AI/ML research artifacts [62] and datasets [21] are available online, allowing full reproducibility of the described models and reported findings.

Explanation. We first reproduced the black-box model (*i.e.,* 1D-CNN) and the results presented in [62, Table VI] for classifying VPN vs. Non-VPN traffic. Next, we used TRUSTEE to extract a DT from the black-box 1D-CNN model (Figure 2) and note that due to the small tree sizes, there was no need for TRUSTEE to apply the Top-k Pruning method. To assess how well the extracted DT reproduces the black-box model, we used it to classify the test cases from [62] and compared the results with the classification from the black-box, measuring precision, recall, and F1. To our surprise, this simple and small white-box model accurately reproduced all black-box decisions, achieving a perfect F1-score.

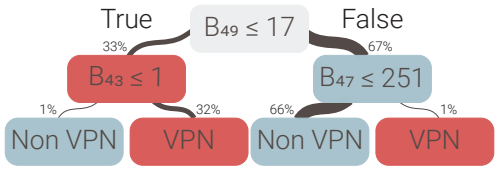

Figure 2: Decision tree for 1D-CNN model. The percentage of samples that follow each branch is presented above each node. Line widths are proportional to the percentage of samples.

Correctly interpreting this extracted DT requires understanding the structure of the input data. Because the DT makes a decision based only on three bytes in the initial segment of each input sample (*i.e.,* bytes B_{49}, B_{43}, and B_{47}), we analyzed samples of VPN and Non-VPN test cases to uncover the "meaning" of those bytes. Figure 3 shows a schematic view of the first 80 bytes of actual input data used in [62]. We notice that each input sample consists of an initial set of bytes representing PCAP metadata, Ethernet header, and IP header. Importantly, none of these initial bytes say anything about actual VPN or Non-VPN traffic.

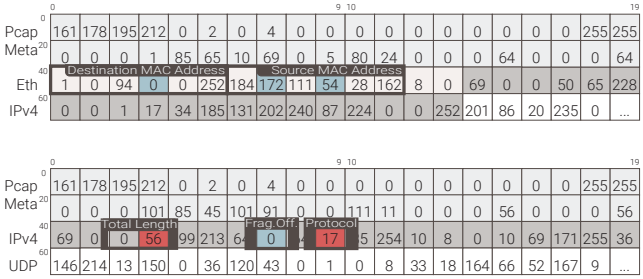

Figure 3: First 80 bytes from the training dataset.

Upon further scrutiny of the public dataset [21], we noticed that Non-VPN traffic samples always contain Ethernet headers while roughly 90% of the VPN traffic samples do not (Figure 3). Thus, if B_k denotes the byte in position k, then for $k \geq 40$, there is a misalignment in the features of the two types of traffic, resulting in completely different semantics for the byte k. In Figure 2, we see that the DT uses feature B_{49} as the splitting criterion at the root node. Due to the feature misalignment, B_{49} is the IPv4 protocol field in VPN samples or the fourth byte of the Ethernet source address in Non-VPN samples. Because the VPN traffic in the dataset uses either UDP ($B_{49} = 17$) or TCP ($B_{49} = 6$), the root node of the DT splits almost all the samples by comparing the IP protocol field in the VPN traffic with a random byte of the Ethernet addresses of the machines used to generate the Non-VPN traffic trace, making feature B_{49} a classical "shortcut" to classify the traffic. However, the split is not perfect because, coincidentally, two machines used for generating Non-VPN traces had the fourth byte of their Ethernet

Table 2: Case Studies.

Section	Problem	Dataset(s)	Model(s)	Trustee Fidelity	Type of inferred inductive bias
Section 7.2	Detect VPN traffic	Public VPN dataset [21]	1-D CNN [62]	1.00	Shortcut learning
Section 7.3	Detect Heartbleed traffic	CIC-IDS-2017 [55]	Multi-class random forest [55]	0.99	Out-of-distribution samples
Section 7.4	Detect Malicious traffic (IDS)	CIC-IDS-2017 [55], Campus dataset	nPrintML [33]	0.99	Spurious correlations
Section 7.5	Anomaly Detection	Mirai dataset [45]	Kitsune [45]	0.99	Out-of-distribution samples
Appendix D	OS Fingerprinting	CIC-IDS-2017 [55]	nPrintML [33]	0.99	Potential out-of-distribution samples
Appendix D	IoT Device Fingerprinting	UNSW-IoT [57]	Iisy [64]	0.99	Likely shortcut learning
Appendix D	Adaptive Bit-rate	HSDPA Norway [50]	Pensieve [43]	0.99	Potential out-of-distribution samples

source addresses less than or equal to 17 (`54:9f:35:`**`0d`**`:e9:c2` and `2c:44:fd:`**`02`**`:16:ef`).

The left branch of the DT classifies most samples as VPNs. However, to weed out a few remaining samples of Non-VPN traffic, the DT uses feature B_{43}. In this case, B_{43} corresponds to the Total Length IP field in most VPN samples or the fourth byte of the Ethernet destination address in Non-VPN samples. Once again, the black-box model takes a shortcut to distinguish between the two classes. A similar analysis applies to the right branch, which classifies most samples as Non-VPN and uses B_{47} (Fragment Offset in Non-VPN vs. second byte of Ethernet source address in VPN) to weed out the few VPN samples.

Validation. Even though the DT extracted by TRUSTEE is a high-fidelity proxy for the 1D-CNN black-box model, it is unreasonable to expect that a simple 3-node structure encompasses the model's entire decision-making process. We verify this intuition by generating a tampered validation dataset for the black-box model. In particular, we changed bytes 43, 47, and 49 in the VPN samples to mimic random Non-VPN samples. By following the logic of the decision tree branches, the black-box model would mis-classify all VPN samples. The first two rows of Table 3 give the average precision, recall, and F1-score for both classes (VPN vs Non-VPN) for original and tampered datasets. The results show that tampering with only these three features out of 748 had no significant impact on the classification accuracy of the black-box model. However, by performing detective work similar to the one described above, we observed that the black-box model succeeds in finding alternative "shortcuts" that are as easy to identify and explain as the one we described earlier.

Table 3: Accuracy of black-box classifier.

Validation dataset	Avg. Precision	Avg. Recall	Avg. F1
Untampered	0.959	0.956	0.955
Tampered-43-47-49	0.959	0.956	0.955
Tampered-32-to-63	0.889	0.861	0.856
Tampered-0-to-63	0.831	0.757	0.734
Tampered-0-to-127	0.753	0.555	0.398

To further demonstrate that the black-box model described in [62] and claimed to be highly successful in learning to classify encrypted VPN and Non-VPN traffic is not a credible predictor, we tampered with entire ranges of bytes instead of individual bytes. As Table 3 shows, tampering with byte ranges of 32-64, 0-64, and 0-128 makes it increasingly more difficult for the black-box model to identify alternative shortcut predictors, and not surprisingly, the model's performance (*i.e.,* accuracy) gets worse and quickly reaches the point where, without being able to resort to shortcut learning (*i.e.,* randomly altering the first 128 bytes, which is less than 18% of the features), the model's performance becomes comparable to that of flipping a fair coin.

7.3 Detecting Heartbleed Traffic

Problem Setup. We consider the paper [55], which presents the public dataset CIC-IDS-2017 with labeled attack traces and lists publications that rely on this dataset to propose ML-based intrusion detection systems. The dataset contains traces of benign background traffic and 13 different attacks, such as Heartbleed, DDoS, and PortScans. The dataset also includes 78 pre-computed flow features, such as flow duration and mean Inter Arrival Time (IAT). Several research efforts report excellent classification results (e.g., average precision and recall above 99% for all classes) of learning models trained on the pre-computed features of this dataset [13, 20, 23, 55, 65].

Explanation. We again started by reproducing the reported classification results using the pre-computed features from the dataset to train a multi-class Random Forest Classifier to identify the 13 attacks and benign traffic, with a 75%-25% train-test split of the data. We could reproduce the excellent results reported by several publications, but, in doing so, we noticed that the dataset in question is highly imbalanced, having as few as 3 Heartbleed samples and as many as 680,000 DDoS samples in the 25% test split. Hence, we used a Random Over Sampler [36, 40] to produce a balanced training dataset to re-train the Random Forest Classifier and then used TRUSTEE to extract a DT explanation. Without applying our Top-k pruning method, the high-fidelity DT extracted by TRUSTEE from the classifier contained 899 nodes, making it largely impossible to understand the decision-making process of the black-box model. However, when running TRUSTEE with the Top-k Pruning method and setting $k = 3$, we obtain the small-sized and therefore inherently manageable DT shown in Figure 4.

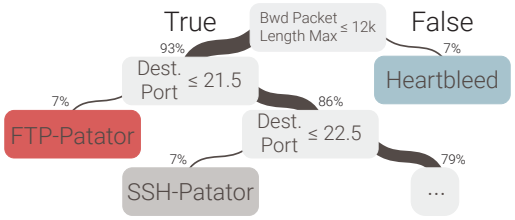

Figure 4: Decision tree for Random Forest Classifier.

Despite the likely shortcut the model takes by using TCP ports to classify SSH and FTP-Patator attacks, the root node of Figure 4 shows that the black-box model correctly classifies all samples of Heartbleed attacks based only on the maximum packet size of the victim server responses (*i.e.,* "Bwd Packet Length Max"). In Heartbleed, an attacker sends a TLS heartbeat message with a value in the size field that is bigger than the message. A vulnerable server responds with a message with a size equal to the value specified in the size field and reviews information stored locally in its memory [22].

Prompted by this observation, we further inspect the DT to identify other features that appear as the most dominant features after we remove the "Bwd Packet Length Max" feature from the dataset. The results showed that the total backward inter-arrival time (*i.e.,* "Bwd IAT Total") also almost perfectly splits all Heartbleed samples. The distributions displayed in the *trust report* for both features (Figure 5) reveal a very telling pattern. To understand this behavior, we inspected the PCAP files and noticed that the TCP connections of the Heartbleed attacks were never closed between heartbeat messages, resulting in high values for the features "Bwd IAT Total" and "Bwd Packet Length Max".

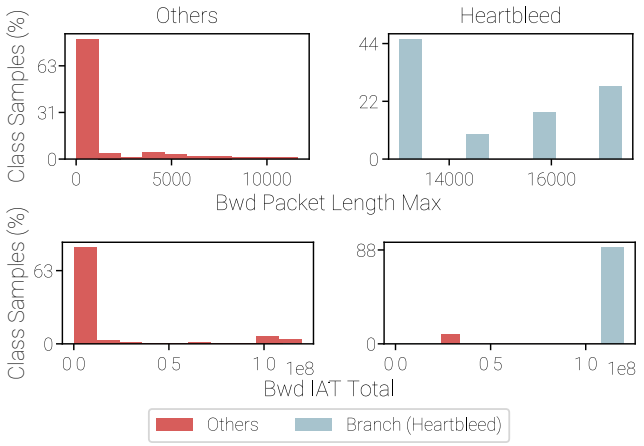

Figure 5: Data distribution of feature "Bwd Packet Length Max" (top) and "Bwd IAT Total" (bottom) comparing values in the Heartbleed class to all Others.

Validation. Considering that the dataset contained just one obvious pattern for the Heartbleed attack, it is not surprising that classifiers trained on this dataset have high accuracy when tested with i.i.d. samples. However, to demonstrate that a model is credible and generalizes as expected in deployment scenarios, we need to validate it with alternate but realistic test cases, *i.e.,* o.o.d. samples. We generated 1,000 new test cases of Heartbleed attacks using the same tool described in [55], but we closed the connection after the heartbeat request triggered a response with compromised data. This change resulted in Heartbleed flows with much smaller backward total IAT but with similar backward maximum packet length, as we use the same packet sizes as for the original trace. We then evaluated the Random Forest Classifier using the newly generated Heartbleed flows as test data. Table 4 shows that with just a simple change in the attack pattern, the classifier could not correctly classify a single one of the 1,000 new Heartbleed attacks, resulting in an F1-score of 0. This experiment demonstrates that the considered black-box learning model overfits on the i.i.d. cases, is not a credible predictor of realistic o.o.d. cases, and does not learn anything that reflects what readily available domain knowledge tells us about Heartbleed attacks.

Table 4: Black-box classifier's accuracy.

Class	Precision	Recall	F1
Heartbleed (i.i.d.)	1.000	1.000	1.000
Heartbleed (o.o.d.)	0.000	0.000	0.000

7.4 Inferring Malicious Traffic for IDS

Problem setup. We consider the paper [33], which proposes nPrint and the stable bit-level representation of network packets for automatically training learning models using AutoML [24]. The idea is to use a sequence of ordered features with values -1, 0, or 1 where each feature represents a bit of a set of pre-established protocol headers. The value -1 represents bits that are not present in a packet, while the values 1 and 0 are the actual values of present bits. The paper shows excellent results for an AutoML IDS model (called nPrintML) with 4,480 features trained using raw PCAP files from the CIC-IDS-2017 dataset [55].

Explanation. We successfully reproduced the reported results using the same configurations as those used in [33], obtaining a model with a 0.999 F1-score. To investigate this high-performance model, we used TRUSTEE (with $k = 4$ for our Top-k Pruning method) to extract a high-fidelity (0.999) DT and show the top-4 branches in Figure 6. We can see that the top nodes rely on bits of the IP TTL field of the packets to separate the Benign class from the others. To understand the reason behind this observation, we inspect the description of the setup used to generate the CIC-IDS-2017 dataset. While all attacks were generated using hosts outside of the network in which the dataset was collected, the benign traffic was from hosts inside the network, creating a strong correlation between the packets' TTL value and traffic type. Also, most attacks were generated by a host running Kali Linux, which sets the initial value for TTL to 64 (*i.e.,* 00**1**00000). Similarly, the DDoS attack traffic was generated using a host running Windows 8.1, which sets the initial TTL value to 128 (*i.e.,* 0**1**000000). This setup where the traffic was generated makes it easy for the model to separate all DDoS attacks using only the second and third most significant bits of the TTL field.

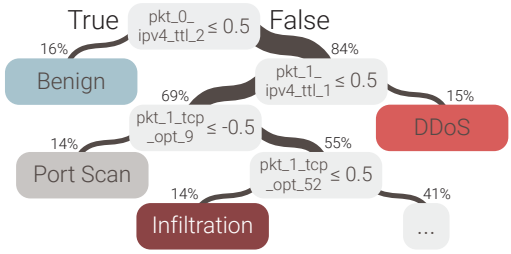

Figure 6: Decision tree for nPrintML IDS model.

We used the extracted DT to further investigate the model's behavior. We iteratively removed (assigned -1 to) the bits of the TTL field and other prominent features from the nPrint representation and retrained the model on the same dataset until the single *tcp_opt* field remained, representing bits of options of the TCP header. Given only these bits, the black-box nPrintML model still separates the attacks in the CIC-IDS-2017 dataset almost perfectly, reaching a F1-score of 0.990. In these cases, the DT explanations produced by TRUSTEE showed that the model still used single bits or packets to divide the traffic perfectly. These experiments demonstrate that the model succeeds in exploiting spurious correlations in the dataset, finding shortcuts due to the vast feature space where each bit is a feature. This issue is also known as the "curse of dimensionality" [53] and concerns cases where a model faces a high-dimensional feature space (*e.g.,* 4,480 features per sample in the case of nPrintML IDS) and not a diverse and dense enough data distribution to avoid occurrences of spurious correlations, which in turn a model can exploit to learn various shortcuts.

Validation. To examine the ability of the nPrintML IDS model to generalize to other deployment environments, we deployed the Suricata Intrusion Detection System [26] in a campus network and mirrored all the traffic before the firewall to produce a real-world dataset of network attacks. We captured about 12 hours of user traffic and the associated Suricata IDS alerts (see Appendix A for details). We found 1,344 flows of DoS attempts. Also, we randomly sampled 1,366 port scan flows (out of 9 million) and 1,337 flows that didn't trigger any alert, which we labeled as benign traffic. Finally, we used nPrint to create a test dataset from that traffic and validated the trained model of [33]. Table 5 shows the classification results of the model for the trace of our campus network.

Table 5: Accuracy for black-box model trained in [33] and tested with traffic captured in our campus network.

Class	Precision	Recall	F1
Benign	0.653	0.806	0.722
DoS	0.000	0.000	0.000
Port Scan	0.120	0.143	0.130
Average	0.256	0.315	0.282

We notice that the model classified most of the traffic as *benign*, a few samples as port-scan attacks, and none as DoS attacks. While we did not expect the model to generalize to real-world settings, we were intrigued that it correctly classified some port scans. Inspecting the decision presented in Figure 6, we can see that the ancestor of the Port Scan node splits most port scan attacks by checking $pkt\_1\_ipv4\_opt\_9 \leq -0.5$. Since the nPrintML model builds its feature vector using the first five packets of a flow (896 features for each packet and 4,480 in total), when a flow has fewer than five packets, it fills the remaining features with -1 values. Hence, to identify port scan attacks, the nPrintML model simply recognizes the absence of the second packet of the flow. To confirm this hypothesis, we carefully investigated the dataset published by the authors of nPrint [33] and noticed that most of the port scans in their dataset have only 1 SYN packet from the attacker to the target (differently from the original PCAPs in [55]). Thus the simple rule that the second packet of a flow is missing would be enough to find all port scans. However, in the case of our campus network traffic, most port-scan attacks also contain a second packet, which prevented the black-box model from classifying this type of traffic. This second packet is a TCP RST packet that attackers send to prevent the target from triggering the TCP SYN Cookie protection used to deal with TCP SYN flooding attacks.

7.5 Anomaly Detection for Mirai Attacks

Problem setup. We analyzed the paper [45], where the authors present Kitsune, an unsupervised ML classifier for anomaly detection. Kitsune comprises an ensemble of auto-encoders and neural networks and solves a regression problem in practice. It receives a set of 115 statistical features (*e.g.,* mean and standard deviation), calculated incrementally for a stream of packets for different levels of aggregation (*e.g.,* by source MAC and IP addresses). It outputs the Root Mean Squared Error (RMSE) as an anomaly score by reconstructing the input features from the ensemble output. Kitsune is trained for some time under normal traffic conditions before moving to an execution phase to detect anomalies. The larger the RMSE, the bigger the anomaly detected by Kitsune. Hence, the authors propose that operators use a threshold-based approach calculated on the training data to detect an anomaly.

Kitsune relies on dampened incremental statistics over time windows, where all features are calculated based on *weights*. The *weight* feature corresponds to the current packet count multiplied by a decay factor so that the weight of older features decreases over time, akin to a sliding window. Kitsune uses a set with five different time windows (100ms, 500ms, 1.5sec, 10sec, and 1min, represented by a variable λ = 5, 3, 1, 0.1, 0.01, respectively) for which the same 23 features are calculated for each time window, resulting in 115 features. While the work described in[45] applies Kitsune to several anomaly detection use cases, a recent study [4] pointed out potential problems with one of these use cases (*i.e.,* the Mirai attack) and prompted us to use TRUSTEE to scrutinize Kitsune's proposed ML model for the Mirai attack.

Explanation. We first executed the Mirai attack-specific experiments described in [45] and were able to reproduce the results reported [45]. The Mirai trace that Kitsune uses for training and evaluation consists of 120 minutes (≈760k packets) of a synthetically generated attack in a network with nine IoT devices, in which the first 70 minutes (≈120k packets) consist of benign traffic and the remaining 50 minutes (≈640k packets) have anomalous traffic. Kitsune is trained on the first 50 minutes of the trace and evaluated on the remainder of traffic. For benign traffic, the largest RMSE computed by Kitsune was approximately 6.9, but for anomalous traffic, this value went up to 14 RMSE. We generated a balanced subset of 300k packets, split between benign and anomalous packets, and used TRUSTEE to extract a high-fidelity (0.99) DT from Kitsune. As Kitsune works as a regression problem, we measure fidelity for this use case as the R-squared value between Kitsune's predictions and those obtained by the DT explanation. Using TRUSTEE with its built-in Top-k Pruning method and setting k = 3 results in the small DT explanation that is shown in Figure 7 and achieves 0.94 fidelity compared to Kitsune.

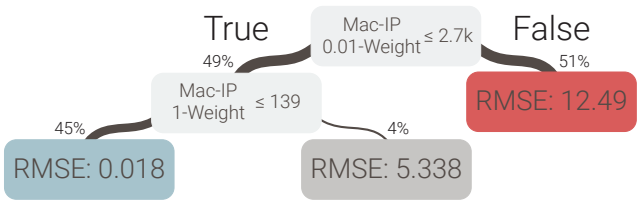

Figure 7: Decision tree for Kitsune Mirai model.

The resulting DT explanation shows that the most prominent features Kitsune uses to determine an anomaly are the *weights*, aggregated by source MAC and IP addresses and associated with two different time windows: 0.01 (1min) and 1 (1.5sec). That is, Kitsune relies mainly on the volume of packets per time frame to determine if an attack is underway. An infected device suffering from a Mirai attack [28] exhibits three main traffic behaviors: *(i)* scanning the network for other vulnerable IoT devices; *(ii)* communicating with the Command and Control (C&C) server, and *(iii)* launching a volumetric DDoS attack from the IoT devices to a target server (usually outside of the infected network). However, the Mirai attack in the synthetic trace used in Kitsune mixes two of these behaviors: a volumetric scan of the infected IoT network with a flood of ARP requests (about 6x times the amount of packets per second compared to the benign traffic, as shown by top-left plot in Figure 8) and a

DDoS attack to the target server. This pronounced difference in volume between benign and attack traffic makes it easy for Kitsune to detect anomalous behavior based on traffic volume alone and corroborates the findings in [4] where a simple Boxplot method is shown to achieve a performance very similar to that of the complex Kitsune. However, as pointed out [4], this difference between malicious and benign traffic for this portion of the Mirai attack is unlikely to be this pronounced in traces collected from real-world networks.

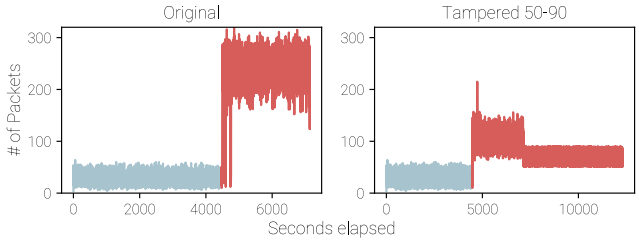

Figure 8: Packets per second for original Mirai trace from Kitsune and tampered trace. Blue segments represent benign traffic and red segments represent traffic with malicious activities (*i.e.,* benign plus attack).

Validation. To validate the DT explanation that TRUSTEE generated, we tampered with the original Mirai trace used in [45]. We modified the attack portion of the original trace by spacing out ARP requests from the Mirai-infected devices so that the number of packets per second (pps) would not cross a random threshold from a given range of specified limits. In particular, by considering the ranges *(i)* from 10 to 50 pps; *(ii)* from 30 to 70 pps and *(iii)* from 50 to 90 pps, we obtained three distinct tampered traces with different volumes of attack traffic (Figure 8 shows the original and one tampered trace). We changed neither the order in which the packets appeared nor the timestamps of ARP responses to avoid interfering with established RTTs. We ran Kitsune for each of these traces, using the same amount of training samples. Figure 9 (left) shows the results for each of the traces' first 200k packets in the execution phase of Kitsune. On the right side of Figure 9, we also compare the expected RMSE (produced by Kitsune in the original trace) and the predicted RMSE for each tampered trace. The diagonal line (in red) represents the optimal outcome between expected and predicted RMSE. Hence, the more dots are closer to the line, the less impact our tampering had on the predicted outcome.

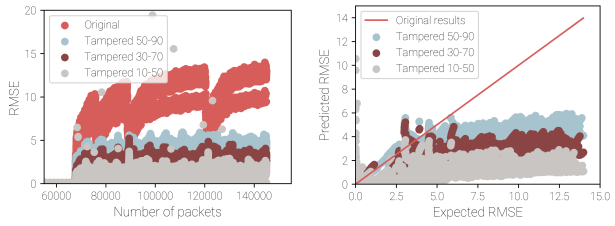

Figure 9: Kitsune execution-phase predicted RMSE results for first 200k packets from original and tampered traces.

The results clearly show that the RMSE values produced by Kitsune depend highly on the volume of the attack traffic encountered, diminishing as the volume decreases, all the way within the values generated for the benign traffic. However, we did notice that our tampering with the original traces made Kitsune produce outliers of RMSE for otherwise benign traffic. While we cannot be sure of the reason for these outliers, since all features calculated by Kitsune depend on the *weight* for each time window, we believe that the changes we made to the attack traffic affected the feature values for the underlying benign traffic. This experiment demonstrates that the Mirai use case from Kitsune is vulnerable to o.o.d. samples, similar to the Heartbleed use case (Section 7.3). A simple but realistic change to the Mirai attack pattern made it impossible for Kitsune to accurately detect anomalous behavior. Finally, while our observations point to problems with Kitsune's ability to detect Mirai attacks, they do not imply that Kitsune is unable or unfit to detect other attacks and problems if it uses training data of representative real-world scenarios.

8 ABLATION STUDY

In this section, we evaluate key design choices we made for TRUSTEE and that we motivated in Section 4.2.

Data augmentation and optimizing for fidelity. We first assess the impact of data augmentation (Line 11 in Algorithm 1) on the size and fidelity of the DT explanations generated by TRUSTEE and at the same time consider the impact of using accuracy (*i.e.,* how well the DT classifies the data) rather than fidelity (*i.e.,* how well the DT mimics black-box classifications) as the optimization goal for TRUSTEE. To this end, for each of the three use cases described in Section 7, we use TRUSTEE to extract DT explanations for four different settings (*i.e.,* with and without data augmentation, using either accuracy or fidelity), with all four settings using the same set of hyperparameter values. The results are shown in Figure 10 where the top plot depicts the (normalized) DT size and the bottom plot shows fidelity. We observe that in cases of small-sized extracted DTs (*e.g.,* maximum tree size for VPN vs. NonVPN and nPrintML IDS is 7 nodes and 47 nodes, respectively), data augmentation is not necessary. However, for extracted DTs that are more complex (*e.g.,* maximum tree size for Heartbleed is 1,491 nodes), the data augmentation step results in a significant reduction in DT size (roughly 20%, and especially for imbalanced datasets) and also improves the DT's fidelity (although only slightly, about 2-3%). In terms of optimizing for fidelity vs. accuracy, we observe no significant differences, mainly because all the analyzed use cases have excellent accuracy to start with. However, we expect that for models that have lower accuracy, optimizing for fidelity may help end users identify reasons for why the model accuracy is low.

Pruning methods. We next evaluate and compare the trade-offs between fidelity and complexity of the DT explanations that TRUSTEE generates when using a tree pruning method other than our proposed Top-k Pruning method (Line 14 in Algorithm 1). In particular, we consider the three pruning methods mentioned in Section 5.1: Max Leaves (pre-pruning), Max Depth (pre-pruning) and CCP (post-pruning). Figure 11 (top) depicts the results for the Heartbleed use case and shows the number of branches (x-axis) and fidelity (y-axis) that are achievable by each of these three methods as well as by our Top-k Pruning method. We observe that except for the Max Depth method, all other methods show similar performance, with the two post-pruning methods (*i.e.,* CCP and Top-k Pruning) outperforming the competitive pre-pruning method Max Leaves, especially for high-fidelity DTs (*e.g.,* fidelity of 0.9 and above). In view of such minimal differences in their overall performance, choosing between CCP and Top-k Pruning boils down to practical considerations. In particular, while the CCP method relies on an implicit parameter α to

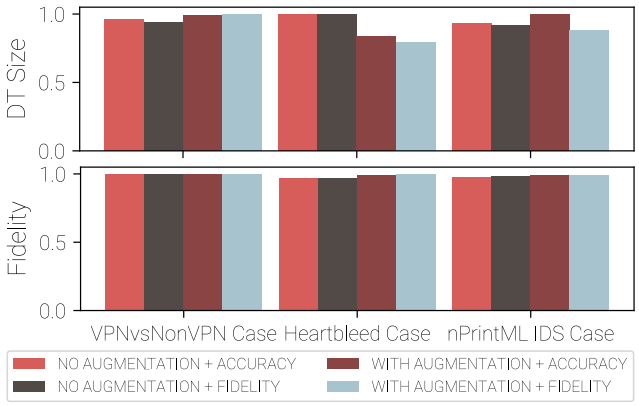

Figure 10: Ablation study results for data augmentation and optimization for fidelity/accuracy.

determine how much post-pruning is necessary, our Top-k Pruning method gives end users direct control by means of the parameter k that explicitly reflects the amount of effort an end user is willing or capable to spend inspecting and analyzing TRUSTEE's output.

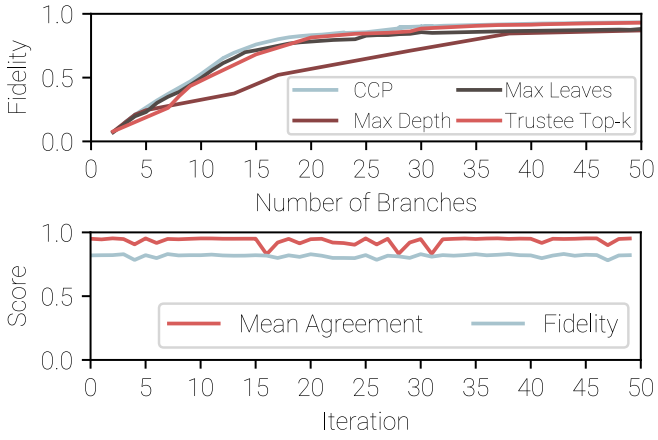

Figure 11: Ablation study results: pruning methods for Heartbleed use case (top), and model stability for Heartbleed use case with Top-10 Pruning (bottom)

Model stability. The last design choice we evaluate concerns the inclusion of an outer loop as part of Algorithm 1 (Lines 4-16). Given that TRUSTEE only analyzes a subset of the input data to generate its output in the form of a DT explanation, it is fully expected that running TRUSTEE under identical conditions (*i.e.,* same set of hyperparameter values) multiple times will result in different DT explanations. However, for an end user to trust the output of TRUSTEE, it should be the case that the different DT explanations are stable in the sense that they make in general identical decisions when presented with the same input samples. We quantify this stability aspect of the output of TRUSTEE by using the notion of agreement between DTs that measures how often the DTs will make the same decision for the same input data. To examine this aspect of TRUSTEE, we consider the Heartbleed use case and ran TRUSTEE (with S=1, thus effectively disabling the outer loop; number of samples M = 593,123 (*i.e.,* 30% of D_0); N = 50; and k = 10) a total of 50 different times. The results are presented in Figure 11 (bottom) and show an overall high mean agreement and fidelity for each of the resulting 50 different DT explanations. However, in a few cases (*e.g.,* iterations 16, 28, 31), the mean agreement of the obtained DT explanations is as low as about 80%. This observation motivated us to include the outer loop in Algorithm 1 that ensures that TRUSTEE outputs a DT explanation that has been selected so as to avoid obviously "bad" (*i.e.,* low mean agreement) and possibly misleading DT explanations for the given black-box model.

9 CONCLUSIONS AND DISCUSSIONS

In this paper, we present TRUSTEE, a new framework that enables end users of ML-based solutions to gauge their trust in the black-box models that underlie these solutions. To demonstrate how TRUSTEE works in practice, we consider several use cases of published ML-based solutions from the existing literature, examine whether end users can trust them, and discuss our findings and lessons learned.

First, we emphasize that our TRUSTEE-based analyses of the considered use cases rely critically on the work of researchers who have made their ML-related artifacts publicly available. In recent years, the scientific community and the network research community, in particular, have argued strongly for more reproducibility [5, 48], and we second this effort. However, for the time being, network security researchers interested in using ML have to accept the lack of open-source datasets and a general reluctance for widespread data sharing due to privacy concerns as *faits accomplis*.

Second, given that the vast majority of published ML models that have been developed for a range of different network security problems are not fully reproducible, our reported findings based on a handful of use cases that are fully reproducible are in no way representative of the existing literature on applications of ML in the field of network security. However, the problematic nature of our findings for the few analyzed use cases should serve as a cautionary tale as far as the popular use of standard ML pipelines in the field is concerned. In this sense, our work contributes to existing efforts that argue for looking at developments in this area with a critical eye (*e.g.,* see [2, 4, 59] and references therein) and identifies specific pitfalls that prevent end users from trusting proposed ML-based solutions and deploying them in production networks.

Last but not least, we purposefully designed TRUSTEE to aid end users' efforts to check whether a given black-box model suffers from the problem of underspecification and can therefore not be trusted. While underspecification is a known and common problem in modern ML pipelines [17], this paper takes a first step towards detecting the presence and identifying the type of underspecification in a given black-box model. However, in the context of TRUSTEE, these detection and identification tasks are currently not automated and depend critically on the help of domain experts who can use TRUSTEE as-is to assert if a given black-box model makes decisions in accordance with existing domain knowledge or is even capable of teaching the domain experts new decision-making strategies. To realize the goal of automating these tasks, much work remains. In particular, we need to involve network operators and security experts in carefully designed user studies for quantitatively assessing their level of trust in a given black-box ML model that drives a proposed ML-based solution for a specific network security problem.

REFERENCES

[1] D. W. Apley and J. Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, 2020.

[2] G. Apruzzese, L. Pajola, and M. Conti. The Cross-evaluation of Machine Learning-based Network Intrusion Detection Systems. *IEEE Transactions on Network and Service Management*, pages 1–1, 2022.

[3] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz. Invariant Risk Minimization. *arXiv preprint arXiv:1907.02893*, 2020.

[4] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and Dont's of Machine Learning in Computer Security. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[5] V. Bajpai, A. Brunstrom, A. Feldmann, W. Kellerer, A. Pras, H. Schulzrinne, G. Smaragdakis, M. Wählisch, and K. Wehrle. The Dagstuhl Beginners Guide to Reproducibility for Experimental Networking Research. *SIGCOMM Comput. Commun. Rev.*, 49(1):24–30, February 2019.

[6] O. Bastani, C. Kim, and H. Bastani. Interpreting Blackbox Models via Model Extraction. *arXiv preprint arXiv:1705.08504*, 2017.

[7] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable Reinforcement Learning via Policy Extraction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 2499–2509, Red Hook, NY, USA, 2018. Curran Associates Inc.

[8] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.*, 13(null):281–305, feb 2012.

[9] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, 2018.

[10] M. Bramer. *Avoiding Overfitting of Decision Trees*, pages 119–134. Springer London, London, 2007.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[12] M. Brundage, S. Avin, J. Wang, et al. Toward Trustworthy AI Development: Mechanisms for Supporting Verifiable Claims. *arXiv preprint arXiv:1705.08504*, 2020.

[13] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever. pForest: In-Network Inference with Random Forests. *arXiv preprint arXiv:1909.05680*, 2019.

[14] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019. Available at https://www.nuscenes.org.

[15] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3D Tracking and Forecasting With Rich Maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. Available at https://www.argoverse.org/.

[16] M. W. Craven and J. W. Shavlik. Extracting Tree-Structured Representations of Trained Networks. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 24–30, Cambridge, MA, USA, 1995. MIT Press.

[17] A. D'Amour, K. Heller, D. Moldovan, et al. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *arXiv preprint arXiv:2011.03395*, 2020.

[18] J. Deng, W. Dong, R. Socher, L. Li a, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[19] A. Dethise, M. Canini, and S. Kandula. Cracking Open the Black Box: What Observations Can Tell Us About Reinforcement Learning Agents. In *Proceedings of the 2019 Workshop on Network Meets AI & ML*, NetAI'19, page 29–36, New York, NY, USA, 2019. Association for Computing Machinery.

[20] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez del Rincón, and D. Siracusa. Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[21] G. Draper-Gil., A. H. Lashkari., M. S. I. Mamun, and A. A. Ghorbani. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP,*, pages 407–414. INSTICC, SciTePress, 2016.

[22] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 475–488. Association for Computing Machinery, 2014.

[23] S. Dwivedi, M. Vardhan, and S. Tripathi. An effect of chaos grasshopper optimization algorithm for protection of network infrastructure. *Computer Networks*, 176:107251, 2020.

[24] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint arXiv:2003.06505*, 2020.

[25] F. Esposito, D. Malerba, G. Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[26] The Open Information Security Foundation. Project Clearwater, January 2018.

[27] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[28] G. Gallopeni, B. Rodrigues, M. Franco, and B. Stiller. A Practical Analysis on Mirai Botnet Traffic. In *2020 IFIP Networking Conference (Networking)*, pages 667–668, 2020.

[29] R. Geirhos, J. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, Nov 2020.

[30] A. Ghorbani, A. Abid, and J. Zou. Interpretation of Neural Networks Is Fragile. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3681–3688, Jul. 2019.

[31] R. Guidotti and R. Ruggieri. On The Stability of Interpretable Models. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[32] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing. LEMNA: Explaining Deep Learning Based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 364–379, New York, NY, USA, 2018. Association for Computing Machinery.

[33] J. Holland, P. Schmitt, N. Feamster, and P. Mittal. New Directions in Automated Traffic Analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 3366–3383, New York, NY, USA, 2021. Association for Computing Machinery.

[34] A. Hussein, M. Gaber Medhat, E. Elyan, and C. Jayne. Imitation Learning: A Survey of Learning Methods. *ACM Comput. Surv.*, 50(2), April 2017.

[35] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville. Paper supplemental material, May 2022. Reproducibility artifacts available at https://github.com/TrusteeML/emperor. The TRUSTEE framework was implemented in a separate Python library for ease of use, available at https://github.com/TrusteeML/trustee.

[36] H. Kaur, H. S. Pannu, and A. K. Malhi. A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions. *ACM Comput. Surv.*, 52(4), aug 2019.

[37] H. Kim and A. Gupta. ONTAS: Flexible and scalable online network traffic anonymization system. In *ACM SIGCOMM Workshop on Network Meets AI & ML*, pages 15–21, 2019.

[38] H. Lakkaraju and O. Bastani. "How Do I Fool You?": Manipulating User Trust via Misleading Black Box Explanations. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, page 79–85, New York, NY, USA, 2020. Association for Computing Machinery.

[39] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec. Interpretable & Explorable Approximations of Black Box Models. *arXiv preprint arXiv:1707.01154*, 2017.

[40] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[41] Z. C. Lipton. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery. *Queue*, 16(3):31–57, June 2018.

[42] S. M Lundberg and S. Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[43] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 197–210, New York, NY, USA, 2017. Association for Computing Machinery. Code available at GitHub repository at https://github.com/hongzimao/pensieve.

[44] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu. Interpreting Deep Learning-Based Networking Systems. In *Proceedings of the Annual Conference of the ACM SIGCOMM*, SIGCOMM '20, page 154–171, New York, NY, USA, 2020. Association for Computing Machinery.

[45] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Network and Distributed System Security Symposium 2018*, NDSS'18, 2018.

[46] C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. Pitfalls to Avoid when Interpreting Machine Learning Models. *arXiv preprint arXiv:2007.04131*, 2020.

[47] National Academies of Sciences Engineering and Medicine. *Implications of Artificial Intelligence for Cybersecurity: Proceedings of a Workshop*. The National Academies Press, Washington, DC, 2019.

[48] B. A. Nosek, G. Alter, G. C. Banks, et al. Promoting an open research culture. *Science*, 348(6242):1422–1425, 2015.

[49] M. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California, June 2016. Association for Computational Linguistics.

[50] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Commute path bandwidth traces from 3G networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118, 2013.

[51] S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *arXiv preprint arXiv:1011.0686*, 2011.

[52] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[53] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, USA, 4rd edition, 2020.

[54] L. S. Shapley. *17. A Value for n-Person Games*, pages 307–318. Princeton University Press, 2016.

[55] I. Sharafaldin., A. H. Lashkari., and A. A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP,*, pages 108–116. INSTICC, SciTePress, 2018.

[56] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu. A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access*, 8:222310–222354, 2020.

[57] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759, 2019.

[58] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[59] R. Sommer and V. Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[60] M. Turek, 2016. Available at https://www.darpa.mil/program/explainable-artificial-intelligence. Accessed on May 25th, 2021.

[61] P. Turney. Technical Note: Bias and the Quantification of Stability. *Machine Learning*, 20(1):23–33, 1995.

[62] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48, 2017. Code available at GitHub repository https://github.com/echowei/DeepTraffic.

[63] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang. Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access*, 6:35365–35381, 2018.

[64] Z. Xiong and N. Zilberman. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA, 2019. Association for Computing Machinery.

[65] H. Zhang, L. Huang, C. Q. Wu, and Z. Li. An effective convolutional neural network based on SMOTE and Gaussian mixture model for intrusion detection in imbalanced dataset. *Computer Networks*, 177:107315, 2020.

Appendices

A ETHICAL CONCERNS

This research does not have any activity that directly involves human subjects. However, packet traces and associated IDS alerts from the campus network may contain personally identifiable information, or PII. Thus, we performed a proper anonymization process for this dataset. Below, we explain the steps taken to conduct our research ethically.

The campus dataset removes any personally identifiable information. All of our data collection and research processes were reviewed and approved by the university's institutional review board (IRB). Our university performs separate reviews to obtain researcher access to administrative data. Since this dataset comes from an operational campus network, the data collection and research processes were also separately reviewed by a committee formed by the institute. The committee, which comprises campus stakeholders and IT experts, has approved to use this dataset for research purposes. All researchers who have or have had access to the campus dataset has gone through proper training before accessing and viewing the dataset. Moreover, all researchers and research projects are reviewed again, every two to three years.

Figure 12 shows the data collection pipeline we instrumented at our campus network. Researchers only had access to the collection server, which is colored in shaded-gray.

Anonymization effort: For the campus dataset, to avoid extracting any privacy-sensitive information, we only collected the five tuples from a packet and the set of IDS rules it triggered. To protect user anonymity, our network operator used a modified ONTAS [37] to anonymize all MAC addresses and local, or campus internal, IP addresses. The program anonymizes privacy-sensitive fields in packets at a programmable data plane at line-rate. Thus, the mirrored campus trace was anonymized *before* the packet trace was even captured at our collection server. We have preserved off-campus, or non-local, IP addresses. Although a remote IP address theoretically can be used to identify a particular off-campus recipient in the Internet, we consider this sufficiently hard to do. Most IP addresses on the Internet belong to large ISPs and corporations (*e.g.,* AT&T, Amazon, Google) or enterprise networks, and are mostly assigned dynamically using the DHCP protocol. Thus, given just a non-campus IP address, it requires considerable effort to be correctly and confidently pinpoint who exactly owned and used the IP address at a given time. Absolutely no effort was made by researchers to pinpoint or identify a particular user using a non-campus, external IP address.

Dataset management: Our campus dataset is stored at a secure infrastructure that is managed by professional IT staff. Only IRB-trained and approved researchers and authorized IT staff members are allowed to access the storage space. The dataset is strictly prohibited from being moved or copied out of the managed infrastructure in general.

Dataset curation: We curated a dataset of real-world traffic from our university campus network. The collected anonymized headers in nPrint format are available alongside the reproducibility assets of the paper [35].

We mirrored to our servers all the traffic of the main campus uplink which destination or source IP addresses were in the university's internal network. We also installed the Suricata IDS 6.0.4 with publicly available rules and selected a subset of rules to be captured, aiming to find the same attacks as were presented in the CIC-IDS-2017 dataset After that, we configured the mirrored traffic to be sent to the Suricata IDS host without any modifications for intrusion analysis, and at the same time, we configured our equipment to truncate the packets leaving only the first 93 bytes and saving the remaining headers. The resulting truncated PCAPs and Suricata's *fast.log* make up the dataset for the analysis.

Using the previously described setup, we captured 12 hours of live network traffic during one of the working days in January 2022. We anonymized all the IP addresses and preprocessed the PCAPs with nPrint according to the original nPrint IDS case preprocessing. For the resulting dataset, we considered any flow that triggered the alert from Suricata IDS as an attack of the corresponding type, so port scans with different settings were all labeled simply as "port scan." For the analysis of nPrintML, we selected a balanced subset of each category that we observed during the traffic capture.

B TRUSTEE EVALUATION

To complement the qualitative comparison we provided in Section 4, we present below a quantitative comparison of fidelity results from existing approaches in the literature to extract decision tree explanation from black-box models. We evaluate all competing approaches using two of the use cases analyzed in Section 7: the VPN-vs-NonVPN Use Case [62], where the black-box model is a 1D-CNN, and the Random Forest Classifier from the CIC-IDS-2017 Use Case [55]. All explanation methods were given the same 200 nodes limit. Table 6 presents the fidelity and size of the resulting explanation from each method.

Since the two analyzed use cases do not rely on RL models, neither VIPER [7] nor Metis [44] could be applied to extract an

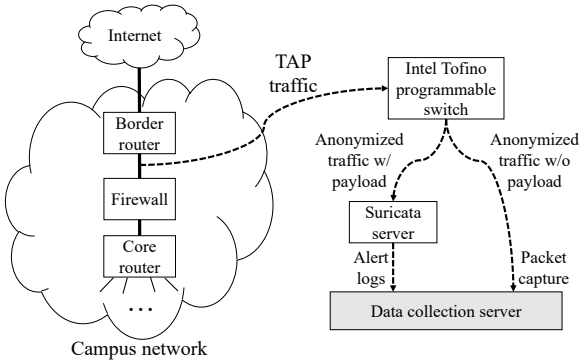

Figure 12: IDS alert and packet trace collection system.

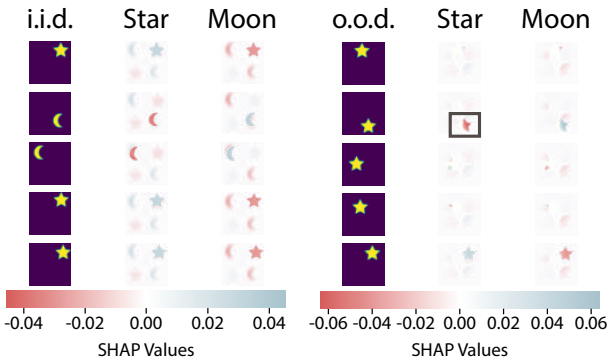

Figure 13: Results from running SHAP on i.i.d. (left) and o.o.d. (right) test samples from shortcut learning toy example. Square on the right mark the cases in which SHAP values show that only position of elements matter, rather than shape.

explanation. In both use cases TRUSTEE was able to achieve a near-perfect fidelity even with the given 200 maximum nodes constraint. While Trepan performed well for the simpler VPN use case, it produces subpar results for the more complex CIC-IDS-2017 trained model. Lastly, dtextract had the worst resulting explanation of all methods compared, producing very low fidelity decision trees.

Note that dtextract's only stopping criterion is the number of nodes. Hence, until the given number of nodes is reached, it will continue adding new nodes to the decision tree, even if it's not necessary, as illustrated by the VPN use case. This behavior also makes the resulting explanation a bad vehicle for identifying the clear shortcut we describe in Section 7 in that specific use case, which Trepan successfully picked up on. In addition, neither Trepan nor dtextract were able to uncover the Heartbleed o.o.d. underspecification problem from the CIC-IDS-2017 use case.

Table 6: Size and fidelity of decision tree explanations produce by existing methods to interpret black-box models for different use cases. Size is measured as the number of nodes in the resulting decision tree. Fidelity is measured as the F1-score between black-box predictions and decision tree predictions.

	VPN vs NonVPN			CIC-IDS-2017		
Method	Max Nodes	Size	Fidelity	Max Nodes	Size	Fidelity
Trepan [16]	200	9	0.99	200	200	0.75
dtextract [6]	200	200	0.55	200	200	0.24
VIPER [7]	-	-	-	-	-	-
Metis [44]	-	-	-	-	-	-
TRUSTEE	200	7	**1**	200	200	**0.99**

C TRUSTEE VS SHAP

To motivate the need for a new global explanation method, we examine an example of shortcut learning presented in [29]. In this toy example (see Figure 2 in [29] for an illustrative diagram), a deep neural network with three fully connected layers classifies images as either moons or stars. The neural network is trained with a biased dataset of images in which the stars are always in the upper-right or lower-left quadrants, while the moons are always in the upper-left or lower-right quadrants. When evaluating the classifier with i.i.d. test samples, it achieves a perfect F1 score, which suggests the neural network learns to distinguish between the shapes of moons and stars.

C.1 SHAP for Moons and Stars Use Case

To verify the credibility of the classifier, we first use SHAP [42] to explain five random i.i.d. test samples. The left plot of Figure 13 shows the SHAP results for each sample: on the left, the original test sample; in the middle, the pixels contributing for or against a 'star' classification, and on the right, the pixels contributing for or against a 'moon' classification. Blue pixels represent pixels contributing to classifying a class, while red pixels represent pixels contributing against that classification. With this explanation, a domain expert would believe the neural network learned each object's correct shape, as all the 'right' pixels of a moon or a star are in blue, and the 'wrong' pixels are in red.

When we test the classifier with o.o.d. samples, generated with stars and moons in random positions, the classifier's accuracy drops to almost the probability of a random coin toss (*i.e.,* F1 score close to 0.5). From the o.o.d. evaluation results, it becomes clear that the classifier did not learn the shape of the objects but rather their positions. By running SHAP on o.o.d. test samples (right plot of Figure 13), a domain expert could detect that the model suffers from underspecification, as only the pixels in the "correct" positions for star and moon are in blue, while stars in the wrong position get highlighted in red. However, o.o.d. are notoriously hard to come by, and relying on such cases to identify underspecification issues is detrimental to the purpose of model explainability.

This simple example illustrates the limitations of local explainability tools like SHAP. When presented with i.i.d. test samples, which is usually the case when testing a classifier, local explanations can easily misguide model developers into believing a model is trustworthy and ready for production. In practice, local explainability tools work better for post-deployment analysis, while a global explanation can give a more 'complete' picture of how the model behaves, allowing a developer to detect such issues before deploying the model.

C.2 TRUSTEE for Moons and Stars Use Case

We start by applying TRUSTEE to extract a decision tree explanation from the trained neural network, shown in Figure 14. Immediately by looking at the decision tree, it's clear that there is some underspecification problem inflicting the trained black-box. The generate decision tree explanation only has 5 nodes (including leaves), and a total of 3 branches. That is, with only two decision splits, we can

achieve a perfect fidelity to the black-box decisions (when testing with i.i.d. samples).

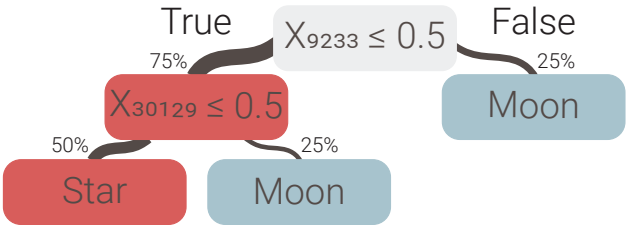

Figure 14: Decision tree explanation extracted from the blackbox neural network model, with fidelity (*i.e.,* F-1 score) = 1. Each node depicts a decision based on the pixel of the images used as input for the black-box model.

By analyzing the decisions made by the generated explanation, we notice that the pixel numbers indicated in the top nodes correspond to pixels in the upper-left quadrant (X_{9233}) and the lower-left quadrant (X_{30129}). So, this explanation tells us that only by looking at two single pixels, which obviously correspond to their positions in the images, the black-box is capable of perfectly distinguishing between moon and star shapes. This toy example perfectly synthesizes how powerful a global explanation tool can be to identify underspecification problems in black-box models. Note, however, that as we discussed in Section 4, our explanation method leverages random sampling to drive explanations to a higher fidelity. This means that, by repeatedly executing TRUSTEE on this black-box model, one would get hundreds of different decision tree explanations with perfect fidelity, as this toy example has numerous intrinsic shortcuts. That is, if we removed the given pixels X_{9233} and X_{30129} from the data samples, the black-box would still find other pixels to perfectly match the desired position of moons and stars, and that would be reflected in the generated decision tree explanations.

D ADDITIONAL RESULTS

We also used TRUSTEE to analyze several additional ML-based models for networking- and network security-related problems that have been reported in the existing literature and are fully reproducible. For brevity, we describe general outcomes of each of them below. These use cases are also available in our supplemental material [35].

OS Fingerprinting. Aside from the IDS use case presented in the nPrint paper [33], we also applied TRUSTEE to scrutinize their OS fingerprinting application of nPrintML. The authors of [33] relied on the same CIC-IDS-2017 published PCAPs and associated OSes from each host in the dataset to build an OS fingerprinting dataset to train nPrintML. The DT explanation TRUSTEE provided largely corroborates the results and claims presented by nPrintML that the model relies on TTL and window size header values to distinguish between Windows, Linux and MacOS OSes. However, we also noted that the black-box model relied on the second most important bit of the TTL header to distinguish Kali Linux OS host machines from regular Linux 3.11 machines. While both OSes have the initial TTL value set to 64, the Kali Linux hosts were positioned outside the network as attackers, and therefore had their TTL value decreased by 2. In the original paper, the authors discard Kali Linux samples because they are underrepresented in the dataset.

IoT Device Fingerprinting. We considered the paper [64], where the authors present a method to deploy in-network ML models using

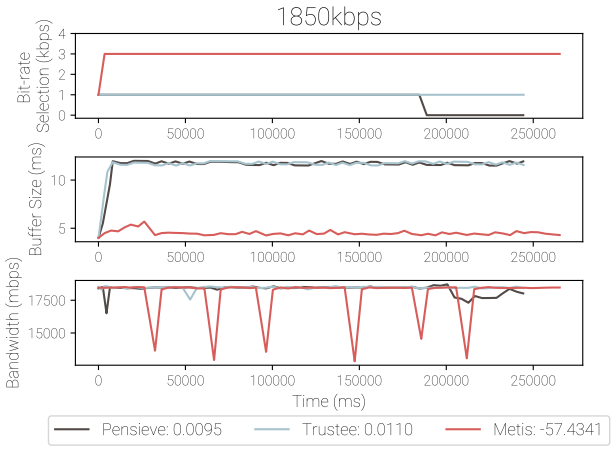

Figure 15: Comparative bit-rate selection results from Pensieve, a Metis-generated decision tree and a TRUSTEE generated decision tree, over a 1850kbps link. The legend shows the overall reward for each method. (Note: Some of these results were obtained using Pensieve's [43] and Metis' [44] reproduction artifacts.

P4. In particular, the authors train a Random Forest Classifier to distinguish between different classes of Internet-of-Things devices (*i.e.,* video, audio, etc), using the UNSW-IoT dataset [57]. From the dataset PCAPs, the authors extract 11 features to train the black-box model, including source and destination TCP and UDP ports but no other identifying headers such as IP and MAC addresses. Applying TRUSTEE to examine the proposed black-box model showed that port numbers were the most prominent features to distinguish between all classes, which obviously makes the model very susceptible to mistakes as port numbers change regularly from device to device, and from manufacturer to manufacturer. Iteratively removing TCP and UDP ports from the data showed a decrease in F1 score from 0.99 to 0.63, with frame length appearing as the most relevant feature, which points to a shortcut learning issue. We do note, however, that the author's goal was not to design the best model possible, but to enable its deployment in-network with P4.

Adaptive Bit-rate. While other works [19, 44] have thoroughly inspected and scrutinized the behavior of the popular Pensieve [43] model to predict adaptive bit-rate, for comparative purposes, we also applied TRUSTEE to produce a DT explanation for Pensieve. Our findings mostly corroborate what previous works [19, 44] have noted: *(i)* Pensieve relies heavily on the previous bit-rate choice and the previous buffer size to choose the best adaptive bit-rate; and *(ii)* Pensieve rarely picks the 1200kbps bit rate or goes higher than 1850kbps. However, note that the DT explanation produced by TRUSTEE achieved a much higher fidelity (*i.e.,* F1 score = 0.90) than the reward-based optimization approach produced by Metis [44] (*i.e.,* F1 score = 0.60), for similar explanation sizes. In an experimental deployment of the DT to predict bit-rate over different link capacities, we observed that the DT produced by TRUSTEE was able to match Pensieve's performance more faithfully than the Metis-generated DT, in particular for link capacities equal and lower than 1850kbps, as illustrated by Figure 15.