Verifiable Learning for Robust Tree Ensembles

Stefano Calzavara
Università Ca' Foscari Venezia
stefano.calzavara@unive.it

Lorenzo Cazzaro
Università Ca' Foscari Venezia
lorenzo.cazzaro@unive.it

Giulio Ermanno Pibiri
Università Ca' Foscari Venezia
giulioermanno.pibiri@unive.it

Nicola Prezza
Università Ca' Foscari Venezia
nicola.prezza@unive.it

ABSTRACT

Verifying the robustness of machine learning models against evasion attacks at test time is an important research problem. Unfortunately, prior work established that this problem is NP-hard for decision tree ensembles, hence bound to be intractable for specific inputs. In this paper, we identify a restricted class of decision tree ensembles, called *large-spread* ensembles, which admit a security verification algorithm running in polynomial time. We then propose a new approach called *verifiable learning*, which advocates the training of such restricted model classes which are amenable for efficient verification. We show the benefits of this idea by designing a new training algorithm that automatically learns a large-spread decision tree ensemble from labelled data, thus enabling its security verification in polynomial time. Experimental results on public datasets confirm that large-spread ensembles trained using our algorithm can be verified in a matter of seconds, using standard commercial hardware. Moreover, large-spread ensembles are more robust than traditional ensembles against evasion attacks, at the cost of an acceptable loss of accuracy in the non-adversarial setting.

ACM Reference Format:
Stefano Calzavara, Lorenzo Cazzaro, Giulio Ermanno Pibiri, and Nicola Prezza. 2023. Verifiable Learning for Robust Tree Ensembles. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623100

1 INTRODUCTION

Machine learning (ML) is now phenomenally popular and found an incredible number of applications. The more ML becomes pervasive and applied to critical tasks, however, the more it becomes important to verify whether automatically trained ML models satisfy desirable properties of interest. This is particularly relevant in the security setting, where models trained using traditional learning algorithms proved vulnerable to *evasion attacks*, i.e., malicious perturbations of inputs designed to force mispredictions at test time, also known as adversarial examples [3, 16, 35].

Unfortunately, verifying the security of ML models against evasion attacks is a computationally hard problem, because verification must account for all the possible malicious perturbations that the attacker may perform. In this work, we are concerned about the security of *decision tree ensembles* [5], a well-known class of ML models particularly popular for non-perceptual classification tasks, which already received significant attention by the research community. Kantchelian et al. [24] first proved that the problem of verifying security against evasion attacks for decision tree ensembles is NP-complete when malicious perturbations are modeled by an arbitrary L_p-norm. In more recent work, Wang et al. [42] further investigated the problem and observed that the existing negative result largely generalizes to the apparently simpler case of decision stump ensembles, i.e., ensembles including just trees of depth one. They thus proposed *incomplete* verification approaches for decision tree and decision stump ensembles, which can formally prove the absence of evasion attacks, but may incorrectly report evasion attacks also for secure inputs. This conservative approach is efficient and provides formal security proofs, however it is approximated and can draw a pessimistic picture of the actual security guarantees provided by the ML model. Complete verification approaches against specific attackers, e.g., modeled in terms of the L_∞-norm, have also been proposed [13, 32]. They proved to be reasonably efficient in practice for many cases, however they have to deal with the NP-hardness of security verification, hence they are inherently bound to fail in the general setting, especially when the size of the decision tree ensembles increases. As a matter of fact, prior experimental evaluations show that security verification does not always terminate within reasonable time and memory bounds, leading to approximated estimates of the actual robustness of the decision tree ensemble against evasion attacks.

Contributions. In this paper we propose a novel approach to the security verification of decision tree ensembles, which we call *verifiable learning*. Our key idea is moving away from the intractable verification problems arising from arbitrary, unconstrained models to rather focus on learning restricted model classes designed to be easily verifiable in practice. In particular:

(1) We identify a restricted class of decision tree ensembles, called *large-spread* ensembles, which admit a security verification algorithm running in polynomial time for evasion attacks modeled in terms of an arbitrary L_p-norm, thus moving away from existing NP-hardness results (Section 3).
(2) We propose a new training algorithm that automatically learns a large-spread decision tree ensemble amenable for efficient security verification. In short, our algorithm first trains a traditional decision tree ensemble and then prunes it to satisfy the proposed large-spread condition (Section 4).

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3623100

$\vec{x}, \vec{z}$	Instances drawn from the feature space $\mathcal{X}$
x_i	i-th component of the vector $\vec{x}$
y	Class label drawn from the set of labels $\mathcal{Y}$
d	Number of features of $\vec{x}$ (i.e., dimensionality of $\mathcal{X}$)
t	Decision tree
n	Number of nodes of a decision tree
T	Tree ensemble
N	Number of nodes of a tree ensemble
m	Number of trees of a tree ensemble
$\vec{\delta}$	Adversarial perturbation
Δ	Norm of an adversarial perturbation
$A_{p,k}$	Attacker based on L_p-norm (max. perturbation k)

Table 1: Summary of notation. In the definitions of n, N, m we assume that the decision tree and tree ensemble we are predicating upon are clear from the context.

(3) We implement our training algorithm and experimentally verify its effectiveness on four public datasets. Our large-spread ensembles are more robust than traditional ensembles against evasion attacks and admit a much more efficient security verification, at the cost of just an acceptable loss of accuracy in the non-adversarial setting (Section 5).

Code availability. We make our code available online (https://github.com/LorenzoCazzaro/Verifiable-Learning-Robust-Tree-Ensembles).

2 BACKGROUND

In this section we review a few notions required to appreciate the rest of the paper. To improve readability, we summarize the main notation used in this paper in Table 1.

2.1 Supervised Learning

Let $\mathcal{X} \subseteq \mathbb{R}^d$ be a d-dimensional vector space of real-valued *features*. An *instance* $\vec{x} \in \mathcal{X}$ is a d-dimensional feature vector $\langle x_1, x_2, \ldots, x_d \rangle$ representing an object in the vector space $\mathcal{X}$. Each instance is assigned a class label $y \in \mathcal{Y}$ by an unknown *target* function $f : \mathcal{X} \to \mathcal{Y}$. As common in the literature, we focus on binary classification, i.e., we let $\mathcal{Y} = \{+1, -1\}$, because any multi-class classification problem can be encoded in terms of multiple binary classification problems.

Supervised learning algorithms automatically learn a *classifier* $g : \mathcal{X} \to \mathcal{Y}$ from a *training set* of correctly labeled instances $\mathcal{D}_{train} = \{(\vec{x}_i, f(\vec{x}_i))\}_i$, with the goal of approximating the target function f as accurately as possible based on the empirical observations in the training set. The performance of classifiers is normally estimated on a *test set* of correctly labeled instances $\mathcal{D}_{test} = \{(\vec{z}_i, f(\vec{z}_i))\}_i$, disjoint from the training set, yet drawn from the same data distribution. For example, the standard *accuracy* measure $a(g, \mathcal{D}_{test})$ counts the percentage of test instances where the classifier g returns a correct prediction.

2.2 Decision Trees and Tree Ensembles

In this paper, we focus on traditional *binary decision trees* for classification [5]. Decision trees can be inductively defined as follows:

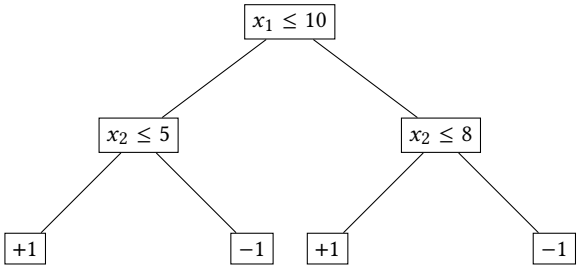


Figure 1: Example of decision tree

a decision tree t is either a leaf $\lambda(y)$ for some label $y \in \mathcal{Y}$ or an internal node $\sigma(f, v, t_l, t_r)$, where $f \in \{1, \ldots, d\}$ identifies a feature, $v \in \mathbb{R}$ is a threshold for the feature, and t_l, t_r are decision trees (left and right child). We just write $\sigma(f, v)$ to represent an internal node when t_l, t_r are unimportant. Decision trees are learned by initially putting all the training set into the root of the tree and by recursively splitting leaves (initially: the root) by identifying the threshold therein leading to the best split of the training data, e.g., the one with the highest information gain, thus transforming the split leaf into a new internal node.

At test time, the instance $\vec{x}$ traverses the tree t until it reaches a leaf $\lambda(y)$, which returns the prediction y, denoted by $t(\vec{x}) = y$. Specifically, for each traversed tree node $\sigma(f, v, t_l, t_r)$, $\vec{x}$ falls into the left sub-tree t_l if $x_f \leq v$, and into the right sub-tree t_r otherwise. Fig. 1 represents an example decision tree of depth 2, which assigns label +1 to the instance $\langle 12, 7 \rangle$ and label −1 to the instance $\langle 8, 6 \rangle$.

To improve their performance, decision trees are often combined into an *ensemble* $T = \{t_1, \ldots, t_m\}$, which aggregates individual tree predictions, e.g., by performing majority voting. We write $T(\vec{x})$ for the prediction of T on $\vec{x}$ and we let N stand for the number of nodes of the ensemble T when such ensemble is clear from the context. For simplicity, we focus on majority voting to aggregate individual tree predictions, assuming that the number of trees m is odd to avoid ties. While ensembles trained using existing frameworks (like sklearn) may use more sophisticated aggregation techniques, our focus on large-spread ensembles trained using a custom algorithm gives us freedom on the choice of the aggregation strategy and majority voting already proves effective in practice. Notable ensemble methods include Random Forest [4] and Gradient Boosting [27].

2.3 Robustness

Classifiers deployed in adversarial settings may be susceptible to *evasion attacks*, i.e., malicious perturbations of test instances crafted to force prediction errors [3, 35]. To capture this problem, the *robustness* measure has been introduced [30]. Below, we follow the presentation in [32].

An *attacker* $A : \mathcal{X} \to 2^{\mathcal{X}}$ is modeled as a function from instances to sets of instances, i.e., $A(\vec{x})$ represents the set of all the adversarial manipulations of the instance $\vec{x}$, corresponding to the possible evasion attack attempts against $\vec{x}$. The *stability* property requires that the classifier does not change its original prediction on some input for all its possible adversarial manipulations.

Definition 2.1 (Stability). The classifier g is *stable* on $\vec{x}$ for the attacker A iff for all $\vec{z} \in A(\vec{x})$ we have $g(\vec{z}) = g(\vec{x})$.

Stability is certainly a desirable property for classifiers deployed in adversarial settings; however, a classifier that always predicts the same class for all the instances trivially satisfies stability for all the attackers, but it is useless in practice because it lacks any predictive power. Robustness improves upon stability by requiring the classifier to also perform correct predictions.

Definition 2.2 (Robustness). The classifier g is *robust* on $\vec{x}$ for the attacker A iff $g(\vec{x}) = f(\vec{x})$ and g is stable on $\vec{x}$ for A.

Based on the definition of robustness, for a given attacker A, we can define the robustness measure $r_A(g, \mathcal{D}_{test})$ by computing the percentage of test instances where the classifier g is robust.

In the following, we focus on attackers represented in terms of an arbitrary L_p-norm, i.e., the attacker's capabilities are defined by some $p \in \mathbb{N} \cup \{0, \infty\}$ and the maximum perturbation k. For fixed p and k, we assume the attacker $A_{p,k}(\vec{x}) = \{\vec{z} \in \mathcal{X} \mid ||\vec{z} - \vec{x}||_p \leq k\}$.

3 EFFICIENT ROBUSTNESS VERIFICATION

We first review results regarding the robustness verification problem for single decision trees (Section 3.1). We then generalize the result to m trees by introducing *large-spread* decision tree ensembles (Section 3.2), which enable robustness verification in $O(N+m \log m)$ time. This is a major improvement over traditional decision tree ensembles, for which robustness verification is NP-complete [24].

3.1 Decision Trees

The robustness verification problem can be solved in $O(nd)$ time for a decision tree with n nodes when the attacker is expressed in terms of an arbitrary L_p-norm [42]. This generalizes a previous result for the L_∞-norm [13]. The key idea of the algorithm is that stability on the instance $\vec{x}$ can be verified by identifying all the leaves that are reachable as the result of an evasion attack attempt $\vec{z} \in A_{p,k}(\vec{x})$; hence, stability holds iff all such leaves predict the same class. This set of leaves can be computed by means of a simple tree traversal. Correspondingly, assuming that $\vec{x}$ has label y, a decision tree t is robust on $\vec{x}$ iff $t(\vec{x}) = y$ and there does not exist any reachable leaf assigning to $\vec{x}$ a label different from y. The algorithm operates in two steps: (1) tree annotation and (2) robustness verification.

3.1.1 Step 1 – Tree Annotation. The first step of the algorithm is a pre-processing operation – performed only once – where each node of the decision tree is annotated with auxiliary information for the second step. The annotations are hyper-rectangles that symbolically represent the set of instances which may traverse the nodes upon prediction. The algorithm first annotates the root with the d-dimensional hyper-rectangle $(-\infty, +\infty]^d$, meaning that every instance will traverse the root. Children are then annotated by means of a recursive tree traversal: concretely, if the father node $\sigma(f, v, t_1, t_2)$ is annotated with $(l_1, r_1] \times \ldots \times (l_d, r_d]$, then the annotations of the roots of t_1 and t_2 are defined as $(l_1^1, r_1^1] \times \ldots \times (l_d^1, r_d^1]$ and $(l_1^2, r_1^2] \times \ldots \times (l_d^2, r_d^2]$ respectively, where:

$$(l_i^1, r_i^1] = \begin{cases} (l_i, r_i] \cap (-\infty, v] = (l_i, \min\{r_i, v\}] & \text{if } i = f \\ (l_i, r_i] & \text{otherwise,} \end{cases} \quad (1)$$

and:

$$(l_i^2, r_i^2] = \begin{cases} (l_i, r_i] \cap (v, +\infty) = (\max\{l_i, v\}, r_i] & \text{if } i = f \\ (l_i, r_i] & \text{otherwise.} \end{cases} \quad (2)$$

The annotation process terminates when all the nodes have been annotated. Note that the complexity of this annotation step is $O(nd)$, because all n nodes are traversed and annotated with a hyper-rectangle of size d.

3.1.2 Step 2 – Robustness Verification. Given an annotated decision tree and an instance $\vec{x}$, it is possible to identify the set of leaves which may be reached by $\vec{x}$ upon prediction in presence of adversarial manipulations.

Let $H = (l_1, r_1] \times \ldots \times (l_d, r_d]$ be the hyper-rectangle annotating a leaf $\lambda(y')$. The minimal perturbation required to push $\vec{x}$ into $\lambda(y')$ is $dist(\vec{x}, H) = \vec{\delta} \in \mathbb{R}^d$, where:[1]

$$\delta_i = dist(\vec{x}, H_i) = \begin{cases} 0 & \text{if } x_i \in H_i = (l_i, r_i] \\ l_i - x_i + \varepsilon & \text{if } x_i \leq l_i \\ r_i - x_i & \text{if } x_i > r_i. \end{cases} \quad (3)$$

Thus, given the instance $\vec{x}$ with label y, it is possible to compute the set:

$$D = \Big\{ ||\vec{\delta}||_p \mid \exists H : dist(\vec{x}, H) = \vec{\delta} \wedge ||\vec{\delta}||_p \leq k \\ \wedge H \text{ annotates a leaf } \lambda(y') \text{ with } y' \neq y \Big\}. \quad (4)$$

In other words, during the visit we find the leaves with a wrong class where $\vec{x}$ might fall as the result of adversarial manipulations by the attacker $A_{p,k}$ and we compute the norms $||\vec{\delta}||_p$ of the minimal perturbations $\vec{\delta}$ to be applied to $\vec{x}$ to push it there. Hence, the tree is robust against the attacker $A_{p,k}$ iff $D = \emptyset$. This computation can be performed in $O(nd)$ time, since we have $O(n)$ leaves and each vector $\vec{\delta}$ with its norm can be computed in $\Theta(d)$ time.

3.2 Generalization to Tree Ensembles

The robustness verification problem is NP-complete for tree ensembles when the attacker is expressed in terms of an arbitrary L_p-norm [24]. Of course, this negative result predicates over *arbitrary* tree ensembles, but does not exclude the possibility that restricted classes of ensembles may admit a more efficient robustness verification algorithm. In this section we introduce the class of *large-spread* tree ensembles, which rule out the key source of complexity from the robustness verification problem and allow robustness verification in $O(N + m \log m)$ time.

3.2.1 Key Intuitions. The key idea of the proposed large-spread condition allows one to verify the robustness guarantees of the individual decision trees in the ensembles and *compose their results* to draw conclusions about the robustness of the whole ensemble.

To understand why composing robustness verification results is unfeasible for arbitrary ensembles, consider the ensemble T in Fig. 2 and an instance $\vec{x}$ with label +1 such that $x_1 = 11$. Consider

[1] We write $x_i - l_i + \varepsilon$ to stand for the minimum floating point number which is greater than $x_i - l_i$. The original paper [13] uses $l_i - x_i$ rather than $l_i - x_i + \varepsilon$, but this is incorrect because $\vec{\delta}$ identifies the minimal perturbation such that $\vec{x} + \vec{\delta} \in H$, however $x_i + l_i - x_i = l_i \notin (l_i, r_i]$. We also assume here that H is not empty, i.e., there does not exist any $(l_j, r_j]$ in H' such that $l_j \geq r_j$.

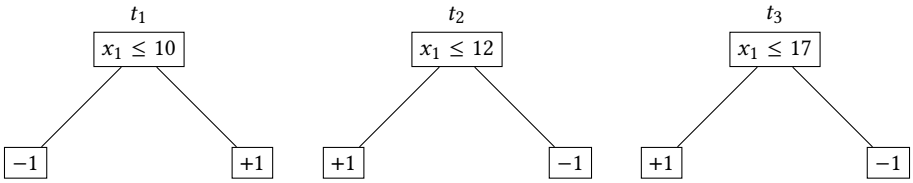


Figure 2: Example of tree ensemble with three decision trees.

the attacker $A_{1,2}$ who can modify feature 1 of at most ±2, then for every adversarial manipulation $\vec{z} \in A_{1,2}(\vec{x})$ we have $z_1 \in [9, 13]$. We observe that the trees t_1 and t_2 are not robust on $\vec{x}$, because there exists an adversarial manipulation that forces them to predict the wrong class -1. However, the whole ensemble T is robust on $\vec{x}$, because $T(\vec{x}) = +1$ and for every adversarial manipulation $\vec{z} \in A_{1,2}(\vec{x})$ we have $T(\vec{z}) = +1$, because either t_1 or t_2 alone is affected by the attack, hence at least two out of the three trees in the ensemble always perform the correct prediction. The example is deliberately simple to show that attacks against two different trees might be *incompatible*, i.e., an attack working against one tree does not necessarily work against the other tree and vice-versa. This implies that the combination of multiple non-robust trees can lead to the creation of a robust ensemble.

The key intuition enabling our compositional reasoning is that interactions among different trees are only possible when the thresholds therein are close enough to each other. Indeed, in our example we showed that there exists an instance $\vec{x}$ which can be successfully attacked in both t_1 and t_2, yet no attack succeeds against both trees at the same time. The reason why this happens is that the thresholds in the roots of the trees (10 and 12 respectively) are too close to each other when taking into account the possible adversarial manipulations: an adversarial manipulation can corrupt the original feature value 11 to produce an arbitrary value in the interval $[9, 13]$, which suffices to enable attacks in both t_1 and t_2. However, none of the attacks against t_1 works against t_2 and vice-versa. Conversely, it is not possible to find any instance $\vec{x}$ which can be attacked in both t_2 and t_3, because for every adversarial manipulation $\vec{z} \in A_{1,2}(\vec{x})$ we have $z_1 \in [x_1 - 2, x_1 + 2]$ and the distance between the thresholds in the trees ($17 - 12 = 5 > 4$) is large enough to ensure that the problem of incompatible attacks cannot exist, because the feature 1 can be attacked just in one of the two trees. For example, if $x_1 = 14$, then only t_2 can be attacked, while if $x_1 = 16$ only t_3 can be attacked; if $x_1 = 15$, instead, neither t_2 nor t_3 can be attacked.

3.2.2 Large-Spread Ensembles. We formalize this intuition by defining the *p-spread* of a tree ensemble T as the minimum distance between the thresholds of the same feature across different trees, according to the L_p-norm. If $\psi_p(T) > 2k$, where k is the maximum adversarial perturbation, we say that T is large-spread.

Definition 3.1 (Large-Spread Ensemble). Given the ensemble $T = \{t_1, \ldots, t_m\}$, its p-spread $\psi_p(T)$ is:

$$\psi_p(T) = \min \bigcup_{\substack{1 \leq f \leq d \\ t,t' \in T, t \neq t'}} \left\{ ||v - v'||_p : \sigma(f, v) \in t \wedge \sigma(f, v') \in t' \right\}.$$

We say that T is *large-spread* for the attacker $A_{p,k}$ iff $\psi_p(T) > 2k$.

A large-spread ensemble T allows one to compose attacks working against individual trees to produce an attack against the ensemble as follows. Assuming $\vec{z}_i = \vec{x} + \vec{\delta}_i$ is an attack against a tree $t_i \in T$ and $\vec{z}_j = \vec{x} + \vec{\delta}_j$ is an attack against a different tree $t_j \in T$, then the large-spread condition guarantees that $\vec{\delta}_i$ and $\vec{\delta}_j$ target disjoint sets of features, i.e., they are orthogonal ($\vec{\delta}_i \cdot \vec{\delta}_j = 0$). Indeed, each feature can be corrupted of k at most, however the same feature can be reused in different trees only if the corresponding thresholds are more than $2k$ away, hence it is impossible for any feature value to traverse more than one threshold as the result of an evasion attack (we formalize and prove this result in the full version [7]).The disjointness condition of attacks implies that $\vec{z} = \vec{x} + \vec{\delta}_i + \vec{\delta}_j$ is an attack working against both t_i and t_j (assuming $||\vec{\delta}_i + \vec{\delta}_j||_p \leq k$), because $t_i(\vec{z})$ and $t_j(\vec{z})$ take the same prediction paths of $t_i(\vec{z}_i)$ and $t_j(\vec{z}_j)$ respectively, which are successful attacks against the two trees. Note that this does not hold for arbitrary tree ensembles, like the one in Fig. 2. Indeed, for that ensemble and an instance $\vec{x}$ such that $x_1 = 11$ the attack against t_1 subtracts 2 from the feature 1 and the attack against t_2 adds 2 to the feature 1, hence the sum of the two attacks would leave the instance $\vec{x}$ unchanged.

3.2.3 Robustness Verification of Large-Spread Ensembles. This compositionality result is powerful, because it allows the efficient robustness verification of large-spread ensembles. The intuition is that – since the ensemble T is large-spread – the minimal perturbations $\{\vec{\delta}_i\}_i$ enabling attacks against the individual trees $\{t_i\}_i$ can be summed up together to obtain a perturbation $\vec{\delta}$ enabling an attack against the whole ensemble. More precisely, let $T' \subseteq T$ be the set of trees in T which may suffer from a successful attack, then:

- If $|T'| < \frac{m-1}{2} + 1$, then the number of trees performing a wrong prediction under attack is too low to identify a successful attack against the whole ensemble.
- If $|T'| \geq \frac{m-1}{2} + 1$, instead, we consider the $\frac{m-1}{2} + 1$ attacks $\{\vec{\delta}_i\}_i$ with the *smallest* L_p-norm. An attack against T is then possible iff $||\vec{\delta}||_p \leq k$, where $\vec{\delta} = \sum_{i=1}^{\frac{m-1}{2}+1} \vec{\delta}_i$.

However, note that the complexity of this algorithm is $O(Nd + m \log m)$ because we annotate each of the N nodes in the ensemble with a hyper-rectangle of size d and we compute the minimum perturbations along with their norms, as explained in Section 3.1. Moreover, to find the perturbations with the smallest norms, we have to sort the pairs $(\vec{\delta}_i, ||\vec{\delta}_i||_p)$ in non-decreasing order of L_p-norm in $O(m \log m)$ time. We now show that the large-spread condition enables a more efficient algorithm, running in $O(N + m \log m)$ time.

3.2.4 Optimization. If the minimal perturbations $\{\vec{\delta}_i\}_i$ are pairwise orthogonal vectors, then the following facts hold.

FACT 1. $||\sum_{i=1}^{q} \vec{\delta}_i||_0 = \sum_{i=1}^{q} ||\vec{\delta}_i||_0$, *if* $\vec{\delta}_i \cdot \vec{\delta}_j = 0, \forall (i,j)$.

FACT 2. $||\sum_{i=1}^{q} \vec{\delta}_i||_\infty = \max_{1 \le i \le q} \{||\vec{\delta}_i||_\infty\}$, *if* $\vec{\delta}_i \cdot \vec{\delta}_j = 0, \forall (i,j)$.

FACT 3. $||\sum_{i=1}^{q} \vec{\delta}_i||_p = (\sum_{i=1}^{q} ||\vec{\delta}_i||_p^p)^{1/p}$, *if* $\vec{\delta}_i \cdot \vec{\delta}_j = 0, \forall (i,j)$.

We introduce the following operator to have a suitable way of referring to the result of the three facts above:

$$\bigoplus_{i=0}^{q} ||\vec{\delta}_i||_p = \begin{cases} \sum_{i=1}^{q} ||\vec{\delta}_i||_0, & \text{if } p = 0 \\ \max_{1 \le i \le q} \{||\vec{\delta}_i||_\infty\} & \text{if } p = \infty \\ (\sum_{i=1}^{q} ||\vec{\delta}_i||_p^p)^{1/p} & \text{if } p \in \mathbb{N}. \end{cases} \tag{5}$$

Fact 1, 2, and 3 imply that we do not actually need to explicitly compute an adversarial perturbation if we just want its L_p-norm, which is exactly our case because we just need to check whether such norm does not exceed k. Since any adversarial perturbation against a large-spread ensemble results from the sum of pairwise orthogonal vectors, we can use Eq. 5 to compute the norm directly from the norms of the orthogonal vectors, i.e., the verification algorithm can operate on scalars rather than vectors, thus reducing its complexity by a d factor.

In light of these considerations, we now revisit the tree traversal from Section 3.1 to show that we can compute for each leaf of the tree just a scalar $\Delta = ||\vec{\delta}||_p$, where $\vec{\delta} = dist(\vec{x}, H)$ and H is the hyper-rectangle which would normally annotate the leaf. Similarly to the linear-time tree visit described in [13] for the L_∞-norm, the idea is to maintain one *global* hyper-rectangle during the visit instead of one hyper-rectangle *per node*. Ultimately, this reduces the time complexity from $O(nd)$ to the optimal $O(n)$, since the hyper-rectangle is not copied from parent to children. The optimized variant of the algorithm is described in the REACHABLE procedure of Algorithm 1. This $O(n)$-time algorithm for arbitrary L_p-norm is, in fact, a combination of the $O(n)$-time algorithm of [13] (which works only for the L_∞-norm) with the generalization to any L_p-norm of [42] (which however runs in $O(nd)$ time).

We implement H as an initially-empty map (e.g., using a hash table): $H_i \in \mathbb{R}^2$ is the entry associated to the i-th feature. If the map does not contain an entry for the i-th feature, then it is implicitly assumed $H_i = (-\infty, +\infty]$. Let $H = (l_1, r_1] \times \ldots \times (l_d, r_d]$ be the state of the hyper-rectangle when visiting node $t = \sigma(f, v, t_1, t_2)$. When moving to a child t_j of t, with $j \in \{1, 2\}$, note that the distance vector $\vec{\delta}$ changes only in its f-th component δ_f, since only the f-th component $(l_f, r_f]$ of the hyper-rectangle H changes. We can therefore update Δ efficiently as follows. Let Δ' and $H' = (l'_1, r'_1] \times \ldots \times (l'_d, r'_d]$ be the perturbation distance and hyper-rectangle associated to any of t's children. Let δ'_f be the quantity defined in Eq. 3. We extend the linear-time algorithm of [13] to an arbitrary L_p-norm by noting that the following is implied by Facts 1, 2, and 3:

$$\text{UPDATE-NORM}(p, \Delta, \delta_f, \delta'_f) = \begin{cases} \Delta - ||\delta_f||_0 + ||\delta'_f||_0 & \text{if } p = 0 \\ \max(\Delta, |\delta'_f|) & \text{if } p = \infty \\ (\Delta^p - |\delta_f|^p + |\delta'_f|^p)^{1/p} & \text{if } p \in \mathbb{N}. \end{cases} \tag{6}$$

By definition, it is clear that UPDATE-NORM is computed in $O(1)$ time. The correctness of the case $p = \infty$ (as also discussed in [13]) follows from the fact that it must be $|\delta'_f| \ge |\delta_f|$, since $(l'_i, r'_i] \subseteq (l_i, r_i]$. In conclusion, we spend $O(1)$ time per node and the time complexity of the whole visit is therefore $O(n)$. Hence, the set D in Eq. 4 is computed in $O(n)$ time rather than $O(nd)$ time as we previously described in Section 3.1. This also lowers the time complexity of the robustness verification for decision trees shown in the ROBUST-TREE procedure of Algorithm 1 to just $O(n)$ rather than $O(nd)$. Since robustness verification for large-spread ensembles builds on the verification algorithm of the individual trees therein, this optimization reduces the complexity of our final algorithm.

Algorithm 1 Optimized robustness verification algorithm for decision trees.

```
1:  function REACHABLE(t, p, k, x⃗, y)
2:      H ← (−∞, +∞]^d
3:      Δ ← 0
4:      return TRAVERSE(t, p, k, x⃗, y, H, Δ)
5:
6:  function TRAVERSE(t, p, k, x⃗, y, H, Δ)
7:      if t = λ(y′) then
8:          if Δ ≤ k and y′ ≠ y then
9:              return {Δ}
10:         return ∅
11:     Let t = σ(f, v, t_l, t_r)
12:     D ← ∅
13:     H*_f ← H_f                                   ▷ copy
14:     δ_f = dist(x⃗, H_f)                           ▷ Eq. 3
15:     H_f ← H*_f ∩ (−∞, v]                         ▷ Eq. 1
16:     δ′_f = dist(x⃗, H_f)                          ▷ Eq. 3
17:     Δ_l ← UPDATE-NORM(p, Δ, δ_f, δ′_f)           ▷ Eq. 6
18:     D ← D ∪ TRAVERSE(t_l, p, k, x⃗, y, H, Δ_l)
19:     H_f ← H*_f ∩ (v, +∞)                         ▷ Eq. 2
20:     δ′_f = dist(x⃗, H_f)                          ▷ Eq. 3
21:     Δ_r ← UPDATE-NORM(p, Δ, δ_f, δ′_f)           ▷ Eq. 6
22:     D ← D ∪ TRAVERSE(t_r, p, k, x⃗, y, H, Δ_r)
23:     H_f ← H*_f                                   ▷ Restore hyper-rectangle
24:     return D
25:
26: function ROBUST-TREE(t, p, k, x⃗, y)
27:     if t(x⃗) = y then
28:         D ← REACHABLE(t, p, k, x⃗, y)
29:         if D = ∅ then
30:             return True
31:     return False
```

3.2.5 Final Algorithm. We conclude this section with Algorithm 2, our robustness verification algorithm for large-spread ensembles, whose correctness is stated in the following theorem and proved in the full version [7]. It follows the description in Section 3.2.3, revised to operate with norms (scalars) rather than vectors.

THEOREM 3.2. *Let $\vec{x}$ be an instance with label y. A tree ensemble T such that $\psi_p(T) > 2k$ is robust on $\vec{x}$ against the attacker $A_{p,k}$ iff* ROBUST$(T, p, k, \vec{x}, y)$ *returns True.*

Algorithm 2 Robustness verification algorithm for large-spread tree ensembles.

```
1:  function Robust(T, p, k, x⃗, y)
2:      if T(x⃗) = y then
3:          return Stable(T, p, k, x⃗, y)
4:      return False
5:
6:  function Stable(T, p, k, x⃗, y)
7:      num_unstable_trees ← 0
8:      Δ⃗ ← [+∞, ..., +∞]                          ▷ Vector of size m
9:      for i ← 1 to m do
10:         D ← Reachable(t_i, p, k, x⃗, y)
11:         if D ≠ ∅ then
12:             Δ_i ← min D
13:             num_unstable_trees ← num_unstable_trees + 1
14:     if num_unstable_trees ≥ (m − 1)/2 + 1 then
15:         Sort Δ⃗ in non-decreasing order
16:         Δ = ⨁_{i=0}^{(m−1)/2+1} Δ_i                ▷ Eq. 5
17:         if Δ ≤ k then
18:             return False
19:     return True
```

Observe that the complexity of Algorithm 2 is $O(N + m \log m)$, where N and m are, respectively, the total number of nodes and trees in the ensemble. Verifying the robustness of the m individual trees in the ensemble and updating vector $\vec{\Delta}$ takes $O(N)$ time thanks to the linear-time Algorithm 1. Afterwards, the algorithm sorts $\vec{\Delta}$ in $O(m \log m)$ time and computes the minimum norm required to attack at least $\frac{m-1}{2} + 1$ trees in $O(m)$ time.

4 TRAINING LARGE-SPREAD ENSEMBLES

We have described an efficient robustness verification algorithm for large-spread ensembles in Section 3. However, traditional decision tree ensembles trained using, e.g., sklearn, do not necessarily enjoy the large-spread condition. Here we discuss possible ideas for training algorithms designed to enforce the large-spread condition and we present a specific solution from the design space.

4.1 Design Space

While reasoning about the design of a training algorithm for large-spread ensembles, we considered different approaches falling in three broad classes:

(1) *Custom ensemble learning algorithms.* Develop new learning algorithms in the spirit of Random Forest [4] or Gradient Boosting [27], designed to constrain the ensemble shape so as to satisfy the large-spread condition. For example, one might train each tree while taking into account the thresholds already present in the previously trained trees, to then remove the training data which might lead to learning thresholds which are too close to the existing ones. Indeed, recall that thresholds are learned from the training data, hence all the possible thresholds are known a priori.
(2) *Training set partitioning.* Pre-compute a partition of the training data so that each decision tree in the ensemble is trained over highly separated instances, thus leading to an ensemble of trees satisfying the large-spread condition. The simplest instantiation of this idea would be partitioning the set of features and train different trees over different subsets of features, so that the large-spread condition is trivially satisfied, but more fine-grained strategies based on instance partitioning would also be feasible.
(3) *Pruning techniques.* Train a standard decision tree ensemble, e.g., using the Random Forest algorithm, and prune it so as to keep only trees satisfying the large-spread condition. A variant of this technique might perform different types of mutations of the available trees to improve the effectiveness of pruning.

Although we consider all these routes to be viable and worth investigating, in this work we decide to prioritize the third class of solutions. Compared to the first class, pruning leads to a range of simple and intuitive solutions, which take advantage of state-of-the-art implementations of existing training algorithms, e.g., those available in sklearn. This simplifies the deployment of an efficient and robust implementation. Moreover, pruning does not necessarily require a massive amount of training data and features, as needed for an effective training set partitioning (second class). In the last part of this section, we also discuss how to leverage feature partitioning to improve the effectiveness of our pruning-based learning algorithm in those settings where a high number of features is available (*hierarchical training*).

4.2 Proposed Training Algorithm

Here we present our training algorithm. We motivate its design, describe how it works and discuss a few relevant aspects of the proposed solution.

4.2.1 Preliminaries. Our problem of interest can be formulated as follows: given a decision tree ensemble T and a size $0 < s \leq |T|$, determine whether there exists an ensemble $T' \subseteq T$ such that T' is large-spread and $|T'| = s$. We refer to this problem as the *large-spread subset* problem for decision tree ensembles. Unfortunately, we can prove that this problem is NP-hard. The proof is provided in the full version [7].

Theorem 4.1. *The large-spread subset problem is NP-hard.*

The theorem implies that it is computationally hard to train large-spread ensembles by pruning when the desired number of trees therein is enforced a priori, which is normally the case because the number of trees is a standard hyper-parameter of ensemble methods. One might argue that this negative result is not a showstopper, because training is performed just once and one might devise efficient heuristic approaches to approximate the large-spread subset problem, however preliminary experiments on public datasets suggest that any training approach which is *purely* based on pruning is likely ineffective in practice. Indeed, we empirically observed on our datasets that traditional random forests trained using sklearn are not directly amenable for pruning, because any two trees in the ensemble already violate the large-spread condition when joined into an ensemble of size two. Our understanding of this phenomenon is that there exist some important features which are pervasively reused across different trees, which often learn the same thresholds,

thus making the identification of a large-spread ensemble unfeasible. Our training algorithm thus integrates a greedy heuristic approach to pruning with a mutation operation, which perturbs thresholds so as to actively enforce the large-spread condition even when it would not be possible by pruning alone.

4.2.2 Training Algorithm. The proposed training algorithm takes as input a training set $\mathcal{D}_{train}$, a number of trees m, a norm p and a maximum perturbation k. In addition to the classic hyper-parameters of tree learning such as tree depth, the algorithm relies on a few specific hyper-parameters: a maximum number of iterations $MAX\_ITER \in \mathbb{N}$, a multiplicative factor $MULT \in \mathbb{N}$ and a real-valued interval $INTV \in \mathbb{R} \times \mathbb{R}$. From a high level point of view, the algorithm operates by training a standard random forest T including $MULT \cdot m$ trees to then select a set of m trees constituting a large-spread ensemble T^*. This is done by a combination of pruning and mutation of the trees in T. After picking a random tree of T to begin with, the algorithm iteratively tries to identify the other $m-1$ trees by means of a greedy approach. The candidate tree t to be inserted in T^* is always the tree in T minimizing the number of *feature overlaps* with T^*, i.e., the number of features violating the large-spread condition in $T^* \cup \{t\}$. If the number of feature overlaps is greater than zero, the ensemble is fixed to enforce the large-spread condition by iteratively removing the overlaps. In particular, let $\sigma(f, v)$ and $\sigma(f, v')$ be two nodes from different trees such that $||v - v'||_p \leq 2k$. We sample a perturbation $\delta \in INTV$, we subtract δ from $\min(v, v')$ and we sum δ to $\max(v, v')$ in the attempt to fix the overlap. Since this change might introduce new overlaps, we then iterate through the ensemble until all the overlaps have been fixed (i.e., the ensemble is large-spread) or the maximum number of iterations *MAX_ITER* have been reached. If all the overlaps of $T^* \cup \{t\}$ have been fixed, i.e., the resulting tree-based ensemble is large-spread, then the extended large-spread ensemble becomes the new large-spread ensemble T^*, otherwise T^* is not extended and the tree t is discarded. Then the algorithm tries to extend T^* with another tree in T, unless T^* has reached the desired number of trees or all the trees in T have been selected for extending the large-spread ensemble. The pseudocode of the training algorithm is presented in Algorithm 3.

4.2.3 Complexity. Recall that each tree has at most n nodes and we fix $MULT$ to be a small constant, e.g., $MULT \in [2, 6]$. TrainLargeSpread calls $O(m)$ times GetBestTree and FixForest. The former function GetBestTree iterates at most $|T| \in O(m)$ times ($t \in T$, line 23) the construction of set *overlaps*. A naive way of building this set is to iterate over all nodes of t (at most n nodes) and compare their thresholds with all the thresholds appearing in the nodes of T^* (at most mn nodes), leading to time $O(mn^2)$ to build one instance of *overlaps*. We observe that it is easy to speed up this step using balanced search trees, but we leave optimizations to further extensions of this work. To sum up, GetBestTree takes $O(m^2n^2)$ and, hence, the $O(m)$ calls to GetBestTree cost overall time $O(m^3n^2)$. Function FixForest iterates MAX_ITER times the `for` loop at line 35. Each iteration of the `for` loop costs $O(1)$ time and there are at most m^2n^2 iterations because the loop iterates over all possible combinations of $\sigma(f, v)$ and $\sigma(f', v')$ belonging to two distinct trees of T^*. Since there are at most mn nodes in T^*, the number of iterations is at most m^2n^2. To this cost, we have to add the MAX_ITER evaluations of "T^* is not large-spread" (line 33); this predicate can be evaluated in $O(|T^*|^2) = O(m^2n^2)$ time by comparing all pairs of thresholds appearing in T^*. We conclude that the running time of the $O(m)$ iterations of FixForest is in total $O(MAX\_ITER \cdot m^3n^2)$. This dominates the running time of the $O(m)$ iterations of GetBestTree, so we conclude that $O(MAX\_ITER \cdot m^3n^2)$ is also the running time of TrainLargeSpread. This cost is paid in addition to the cost of training the standard random forest at line 2.

As noted above, although it is feasible to reduce this complexity using appropriate data structures, we observe that (*i*) training is often performed only once, so any optimization just offers limited

Algorithm 3 Training algorithm for large-spread ensembles

Require: Hyper-param. $MAX\_ITER \in \mathbb{N}, MULT \in \mathbb{N}, INTV \in \mathbb{R} \times \mathbb{R}$

1: **function** TrainLargeSpread($\mathcal{D}_{train}, m, p, k$)
2: $T \leftarrow$ TrainRandomForest($\mathcal{D}_{train}, MULT \cdot m$)
3: $t \leftarrow$ SampleTree(T) ▷ Choose a random tree from T
4: $T \leftarrow T \setminus \{t\}$
5: $T^* \leftarrow \{t\}$
6: $i \leftarrow 1$
7: **while** $i < MULT \cdot m$ and $|T^*| < m$ **do**
8: $i \leftarrow i + 1$
9: $t \leftarrow$ GetBestTree(T, T^*, p, k)
10: $T \leftarrow T \setminus \{t\}$
11: $\overline{T^*} \leftarrow T^* \cup \{t\}$
12: $\overline{T^*} \leftarrow$ FixForest($\overline{T^*}, p, k$)
13: **if** $\overline{T^*} \neq \bot$ **then** ▷ FixForest succeded
14: $T^* \leftarrow \overline{T^*}$
15: **if** $|T^*| = m$ **then** ▷ TrainLargeSpread succeded
16: **return** T^*
17: **else**
18: **return** $\bot$
19:
20: **function** GetBestTree(T, T^*, p, k)
21: $t^* \leftarrow \bot$
22: $min\_f\_overlaps \leftarrow +\infty$
23: **for** $t \in T$ **do**
24: $overlaps \leftarrow \{\sigma(f, v) \in t \mid \exists \sigma(f, v') \in T^* : ||v - v'||_p \leq 2k\}$
25: $f\_ov \leftarrow |\{f \mid \exists \sigma(f, v) \in overlaps\}|$
26: **if** $f\_ov < min\_f\_overlaps$ **then**
27: $min\_f\_overlaps \leftarrow f\_ov$
28: $t^* \leftarrow t$
29: **return** t^*
30:
31: **function** FixForest(T^*, p, k)
32: $iter \leftarrow 0$
33: **while** T^* is not large-spread and $iter < MAX\_ITER$ **do**
34: $iter \leftarrow iter + 1$
35: **for** $t, t' \in T^*,\ \sigma(f, v) \in t,\ \sigma(f', v') \in t'$ **do**
36: **if** $f = f'$ and $||v - v'||_p \leq 2k$ **then**
37: $\delta \leftarrow$ Random($INTV$) ▷ Sample a float in $INTV$
38: **if** $v \leq v'$ **then**
39: $\sigma(f, v) \leftarrow \sigma(f, v - \delta)$
40: $\sigma(f', v') \leftarrow \sigma(f', v' + \delta)$
41: **else**
42: $\sigma(f, v) \leftarrow \sigma(f, v + \delta)$
43: $\sigma(f', v') \leftarrow \sigma(f', v' - \delta)$
44: **if** T^* is not large-spread **then**
45: **return** $\bot$
46: **return** T^*

benefits and is left to future work, and (ii) the number of trees and nodes is often small enough to make a cubic complexity acceptable in practice. As a matter of fact, our experimental evaluation gives evidence about the acceptable empirical efficiency of the proposed training algorithm.

4.2.4 Hierarchical Training. We observe that our training algorithm can fail, in particular when it is not possible to add one tree to the current large-spread ensemble and reduce to zero the overlaps by our mutation routine, i.e., the number of overlaps resulting from adding a tree to the large-spread ensemble is too high. However, we show in our experimental evaluation (see Section 5) that it is possible to train large-spread ensembles of different dimensions after some parameter tuning. In particular, we propose an intuitive and effective technique to mitigate the risks of failures during training. A key insight is that the larger the ensemble is, the more difficult it becomes to avoid violations of the large-spread requirement, because ensembles including many trees also have many thresholds, hence overlaps become harder to avoid. We thus propose a *hierarchical* training approach as follows:

(1) We first partition the set of features in l disjoint subsets and we build l different projections of the training set $\mathcal{D}_{train}$, based on such feature sets.
(2) We train a large-spread ensemble of size $\frac{m}{l}$ on each of the l different training sets using Algorithm 3 and we finally merge all the trained ensembles into an ensemble of m trees.

Note that the final ensemble is indeed large-spread, because each of the merged ensembles ensures the large-spread condition on the trees therein, and trees from different ensembles cannot violate the large-spread condition because they are built on disjoint sets of features. For example, an ensemble of 100 trees can be trained by building 4 disjoint projections of the training data (based on feature partitioning) and training an ensemble of 25 trees on each of them. We empirically observed that this approach may improve the effectiveness of the training process, by enabling the construction of larger ensembles in practice. We report on experiments confirming this observation in the next section.

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup

To show the practical relevance of our theory, we develop two tools on top of it and we prove their effectiveness on public datasets.

5.1.1 Tools. Our first tool CARVE[2] is a C++ implementation of the proposed robustness verification algorithm for large-spread ensembles (Algorithm 2). It takes as input a random forest classifier T, a norm p, a maximum perturbation k and a test set $\mathcal{D}_{test}$ to return as output the robustness score $r_{A_{p,k}}(T, \mathcal{D}_{test})$. CARVE assumes that T is large-spread and implements majority voting as the aggregation scheme of individual tree predictions. Our second tool LSE is a sequential Python implementation of the proposed training algorithm for large-spread ensembles (Algorithm 3). Starting from a training set $\mathcal{D}_{train}$, a number of trees m, a norm p and a maximum perturbation k, it returns a large-spread ensemble T^* of m trees (unless the training algorithm fails by returning $\perp$). The random forest trained before pruning is created using sklearn.

[2]CARVE - CompositionAl Robustness Verifier for tree Ensembles

Table 2: Dataset statistics

Dataset	Instances	Features	Distribution
Fashion-MNIST	13,866	784	50%/50%
MNIST	14,000	784	51%/49%
REWEMA	6,271	630	50%/50%
Webspam	350,000	254	70%/30%

5.1.2 Methodology. Our experimental evaluation is performed on four public datasets: Fashion-MNIST[3], MNIST[4], REWEMA[5] and Webspam[6]. Since Fashion-MNIST and MNIST are datasets associated to multiclass classification tasks and we focus on binary classification tasks in this work, we consider two subsets of them. In particular, for Fashion-MNIST we consider the instances with class 0 (T-shirt/top) and 3 (Dress), while for MNIST we keep the instances representing the digits 2 and 6. The key characteristics of the chosen datasets are reported in Table 2. The chosen datasets are representative for different reasons: Fashion-MNIST, MNIST and Webspam have already been considered in the robustness verification literature [1, 13, 32, 42]; moreover, REWEMA and Webspam are associated with a security-relevant classification task (malware and spam detection, respectively) for which the robustness verification of the employed classifier is critically important. In general, we choose datasets with a high number of features, where it may be useful to train large tree ensembles to reach the best performance. Each dataset is partitioned into a training set and a test set, using 70/30 stratified random sampling.

In our experimental evaluation we make use of two training algorithms to learn different types of classifiers: (i) a majority-voting classifier based on a traditional random forest (RF) trained using sklearn, and (ii) a majority-voting classifier based on a large-spread tree ensemble trained using LSE. Moreover, we consider tree-based classifiers of different sizes: (i) small ensembles with 25 trees of maximum depth 4; (ii) large ensembles with 101 trees with maximum depth 6. We only consider ensembles with an odd number of trees in order to avoid ties in classification.

Robustness verification is then performed using CARVE and SILVA, a state-of-the-art verifier for traditional decision tree ensembles based on abstract interpretation [32]. Note that SILVA can be applied to arbitrary ensembles, while CARVE can only be used on large-spread ensembles. Since SILVA leverages the hyper-rectangle abstract domain for verification, which does not introduce any loss of precision for L_∞-norm attackers but might lead to an over-approximation for generic L_p-norm attackers, we only focus on L_∞-attackers in our comparison. For the sake of completeness, in our evaluation of CARVE we also consider robustness against L_1-attackers and L_2-attackers for large-spread ensembles.

Finally, in our evaluation we consider different perturbations $k \in \{0.0050, 0.0100, 0.0150\}$ for the MNIST, Fashion-MNIST and

[3]https://www.openml.org/search?type=data&sort=runs&id=40996&status=active
[4]https://www.openml.org/search?type=data&sort=runs&id=554
[5]https://www.kaggle.com/code/kerneler/starter-rewema-c5ce57b7-e/input
[6]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

REWEMA datasets, while we assume smaller perturbations drawn from the set {0.0002, 0.0004, 0.0006} for Webspam. We choose different perturbations for the Webspam dataset to be aligned with previous work and to obtain roughly the same decrease in robustness observed on the other three datasets for the considered tree-based classifiers. Indeed, Chen et. al. [12] showed in their experimental evaluation that the certified minimum adversarial perturbation obtained for the Webspam dataset is one order of magnitude smaller than the one obtained for the MNIST dataset, i.e., models trained over Webspam would be too fragile to be usable when tested against larger perturbations.

5.1.3 LSE Setup. Our tool LSE requires the user to specify the value of some additional parameters (described in Section 4.2) with respect to the traditional implementation of the training algorithm for random forests by sklearn. The norm p and the perturbation k depend on the assumed attacker's capabilities, so they do not require a particular tuning. Still, other parameters such as the number of partitions l for the hierarchical training and the maximum number of iterations *MAX_ITER* of the FixForest procedure require some tuning. Indeed, although partitioning the features may enable the training of larger ensembles, a too high number of partitions might negatively affect the accuracy of the resulting large-spread ensemble, because each sub-forest has only a partial view on the set of available features and some patterns may not be learned. In the same way, the maximum number of rounds *MAX_ITER* has an impact on the success of the training procedure, since a minimum number of rounds is required to adjust the thresholds of the ensemble, but a too high number of rounds may modify the thresholds too much and downgrade the predictive power of the model. We perform some experiments in order to assess the influence of these parameters on the success of training a large-spread ensemble and on the accuracy of the resulting model to then pick the best-performing models in our experimental evaluation. For space reasons, we discuss details in the full version [7].

5.2 Accuracy and Robustness Results

In our first experiment we assess whether large-spread ensembles are effective at classification and we analyze their robustness properties. Indeed, the large-spread condition enforced on the ensemble limits the model shape, thus potentially reducing its predictive power with respect to traditional tree ensembles. Since we are not just concerned about accuracy but we target robustness, we also analyze how large-spread ensembles fare against evasion attacks. Our evaluation consists of two parts. We first compare the accuracy and robustness of the large-spread ensembles against traditional random forests of the same size, considering an L_∞-attacker. The robustness of the traditional models is computed using SILVA, since CARVE can only be used for verifying large-spread ensembles. We set a timeout per instance of one second, as in [32]. Then, we use CARVE to verify the robustness of large-spread ensembles against L_1-attackers and L_2-attackers that are not supported by SILVA.

5.2.1 Comparison for L_∞-norm Attackers. Table 3 shows the experimental results of our comparison. Note that the value of robustness may be approximated, since SILVA may not be able to verify robustness on some instances within the time limit; for these cases, we provide lower and upper bounds of robustness, using the ± notation. The results highlight that the large-spread ensembles are *reasonably accurate* and often *more robust* than the random forests of the same size. In particular, the accuracy of the large-spread ensembles is at most 0.03 lower than the accuracy of the corresponding traditional model in the majority of the cases, while the improvement in robustness is at least 0.04 in around half of the cases. This is reassuring, because accuracy was at stake, since the large-spread condition restricts the shape of the ensemble and might be associated to a reduction of predictive power. The increase of robustness is an interesting byproduct of the large-spread condition: since thresholds in different trees are far way, evasion attacks are empirically harder to craft. Observe that the accuracy and robustness values of the large-spread ensembles on the MNIST and Fashion-MNIST test sets show that large-spread models present better performance overall than the traditional ensembles. The accuracy of the large-spread ensembles on these two test sets is usually equal to the one of the traditional ensembles, while the robustness value improves of at least 0.06 in half of the cases, in particular when the largest considered perturbation k is used as the attacker's capability. For example, the robustness of the large-spread ensemble with 101 trees of maximum depth 6 and perturbation 0.0150 is at least 0.22 higher than the robustness of the corresponding random forest, while the accuracy decreases only by 0.04 at most. When the value of the perturbation k is the lowest considered, the results are still positive, since the large-spread ensembles present the same accuracy and a higher robustness than the ones of the traditional ensembles.

We see a slightly different trend in the results for the REWEMA and Webspam datasets: the robustness of large-spread ensembles is always equal to or greater than the robustness of the traditional ensembles, but the gap in accuracy with respect to the traditional ensembles may increase, in particular when considering large adversarial perturbations, which make it harder to enforce the large-spread condition. For example, the large spread ensembles of 101 trees with maximum depth 6 trained on the two datasets present 0.88 and 0.82 robustness with perturbation 0.015 and 0.0006 (respectively, +0.10 and +0.01 than the robustness of the corresponding traditional tree ensembles), but their accuracy is 0.88 and 0.85 (respectively, −0.10 and −0.09 than the accuracy of the traditional tree ensembles). This confirms that an improvement in robustness often occurs at the price of a decrease in accuracy, because of the classic trade-off between accuracy and robustness [31, 38]. Even in these cases though, adopting large-spread ensembles continues to be useful: the accuracy is always way above the majority class distribution, so the model is usable in the non-adversarial setting, while being normally more robust than the traditional counterpart and amenable for efficient security verification. To explain the observed drop in accuracy for large-spread models, we compare the *permutation feature importance* [4] for traditional ensembles and large-spread ensembles to assess which features have more predictive power according to the different models. The analysis is quite interesting. For the REWEMA dataset, it shows that traditional models give significant importance to a few numerical features which are less important for large-spread models; large-spread models, in turn, privilege some categorical / ordinal features which are less important for traditional models. Instead, for the Webspam dataset,

Table 3: Accuracy and robustness measures for traditional and large-spread ensembles. Robustness is computed against $A_{\infty,k}$. We highlight in bold the cases in which the gap between the accuracy and the robustness of the traditional tree-based ensemble and large-spread ensemble is at least of 0.05.

Dataset	k	Trees	Depth	Accuracy		Robustness	
				Traditional	Large-Spread	Traditional	Large-Spread
Fashion-MNIST	0.0050	25	4	0.93	0.92	0.90	0.90
		101	6	0.96	0.96	0.91	0.93
	0.0100	25	4	0.93	0.92	0.86	0.87
		101	6	0.96	0.94	**0.79**	**0.91**
	0.0150	25	4	0.93	0.91	**0.60**	**0.88**
		101	6	0.96	0.92	**0.51 ± 0.01**	**0.89**
MNIST	0.0050	25	4	0.97	0.97	**0.90**	**0.96**
		101	6	0.99	0.99	0.94	0.97
	0.0100	25	4	0.97	0.97	**0.72**	**0.90**
		101	6	0.99	0.99	**0.77 ± 0.02**	**0.97**
	0.0150	25	4	0.97	0.97	**0.64**	**0.83**
		101	6	0.99	0.99	**0.67 ± 0.05**	**0.94**
REWEMA	0.0050	25	4	0.88	0.88	0.85	0.87
		101	6	**0.98**	**0.89**	0.88	0.89
	0.0100	25	4	0.88	0.88	0.83	0.87
		101	6	**0.98**	**0.89**	0.86	0.88
	0.0150	25	4	0.88	0.88	0.83	0.85
		101	6	**0.98**	**0.88**	**0.78**	**0.88**
Webspam	0.0002	25	4	0.90	0.90	0.83	0.87
		101	6	0.94	0.91	0.88	0.90
	0.0004	25	4	0.90	0.89	**0.80**	**0.86**
		101	6	**0.94**	**0.89**	0.85	0.86
	0.0006	25	4	0.90	0.89	**0.78**	**0.85**
		101	6	**0.94**	**0.85**	0.81	0.82

it shows that traditional and large-spread models privilege numerical features with many distinct values. However, the traditional models give also importance to some features with a very skewed empirical distribution towards the value 0, while the large-spread ensembles give more importance to features with scattered values. This motivates why large-spread models sacrifice some predictive power, but show better robustness in general: categorical / ordinal features and, in general, features with more scattered values are harder to target for L_p-norm attackers, because their sparse nature makes them more robust to adversarial perturbations, i.e., larger perturbations are required to actually traverse thresholds and thus affect predictions.

5.2.2 Additional Attackers. Table 4 shows the robustness of the trained large-spread ensembles against different L_p-attackers for $p \in \{1, 2, \infty\}$. As expected, the large-spread ensembles trained on MNIST and Fashion-MNIST are generally more robust against the weakest L_1-attacker and less robust against the strongest L_∞-attacker. Instead, we observe that the large-spread ensembles trained on the REWEMA and Webspam datasets show a different behaviour: the robustness values of the large-spread ensemble models are almost the same for every attacker considered. This is explained by the fact that large-spread models trained over such datasets make a more significant use of categorical / ordinal features and features with more scattered values, as discussed in the previous section. The attacker thus cannot perturb the test instances to cross thresholds of important features for prediction, independently of the chosen L_p-norm. We remark here that the effectiveness of CARVE does not depend upon p: robustness verification is always exact and the complexity of the analysis is independent from p. This motivates why the rest of our evaluation only considers the case $p = \infty$.

5.3 Efficiency of Robustness Verification

We now compare the SILVA and CARVE robustness verification tools along two different dimensions: verification time and memory consumption. For simplicity, we only focus on the verification of large ensembles with 101 trees and maximum depth 6 on the MNIST dataset with $k = 0.0150$. As emerged from the results in Section 5.2, this is a setting where a state-of-the-art approach like SILVA clearly shows its limits: indeed, SILVA could not provide a precise estimate of the robustness of this model (± 0.05). In order to measure the verification time per instance and setting timeouts in the same way for both the tools, we use the GNU commands `time` and `timeout` that measure the elapsed wall clock time. The former command is also used to compute the maximum amount of physical memory

Table 4: Robustness measures for large-spread ensembles against different L_p-attackers.

Dataset	k	Trees	Depth	Robustness		
				$A_{\infty,k}$	$A_{2,k}$	$A_{1,k}$
Fashion-MNIST	0.0050	25	4	0.90	0.90	0.90
		101	6	0.93	0.93	0.94
	0.0100	25	4	0.87	0.88	0.89
		101	6	0.91	0.91	0.93
	0.0150	25	4	0.88	0.89	0.89
		101	6	0.89	0.89	0.91
MNIST	0.0050	25	4	0.96	0.96	0.97
		101	6	0.97	0.98	0.98
	0.0100	25	4	0.90	0.93	0.95
		101	6	0.97	0.98	0.98
	0.0150	25	4	0.83	0.88	0.93
		101	6	0.94	0.95	0.97
REWEMA	0.0050	25	4	0.87	0.87	0.87
		101	6	0.89	0.89	0.89
	0.0100	25	4	0.87	0.87	0.87
		101	6	0.88	0.88	0.88
	0.0150	25	4	0.85	0.87	0.87
		101	6	0.88	0.88	0.88
Webspam	0.0002	25	4	0.87	0.88	0.88
		101	6	0.90	0.90	0.90
	0.0004	25	4	0.86	0.86	0.86
		101	6	0.86	0.86	0.87
	0.0006	25	4	0.85	0.86	0.86
		101	6	0.82	0.83	0.83

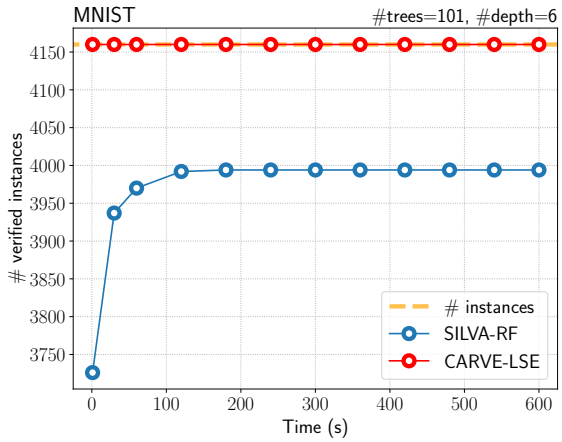
allocated to the verifier. When it is required to set a maximum amount of physical memory that the process can use, we use the Linux kernel feature `cgroup`. All the experiments are performed on a virtual machine with 103 GB of RAM and Ubuntu 20.04.4 LTS, running on a server with an Intel Xeon Gold 6148 2.40GHz.

5.3.1 Time Efficiency. In our first experiment we compare the robustness verification times for traditional tree ensembles using SILVA and the robustness verification times for large-spread ensembles using CARVE. This way, we compare a state-of-the-approach for adversarial machine learning models (i.e., what we would do today) against our custom algorithm designed to take advantage of the large-spread condition (i.e., what we put forward in this paper). In the experiments of Section 5.2, we set the maximum verification time per instance of SILVA to one second. However, SILVA may complete the verification also on more difficult instances if more time is granted, e.g., 60 seconds [32]. In order to perform a fair comparison, we compare how many instances of the MNIST test set can be verified under growing time limits per instance, i.e., from one second to 10 minutes. This methodology allows us to figure out on how many instances the verification is really difficult. Note that the timeout of 10 minutes per instance is already extremely large, since test sets normally include thousands of instances.

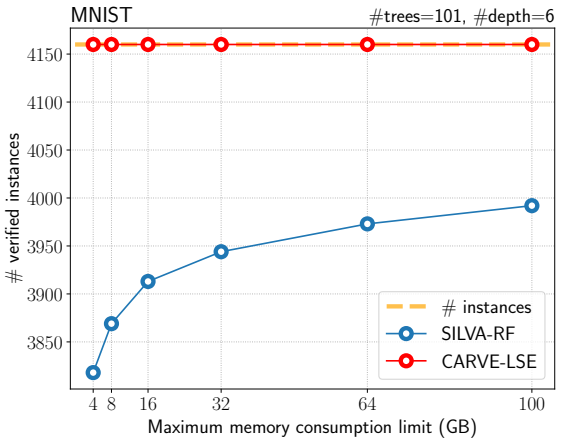
Figure 3a shows the results of our experiment. The plot shows that SILVA is not able to verify the robustness of the traditional tree ensemble on 434 instances in less than one second and on 190 instances in less than one minute, providing just approximate robustness estimates with an uncertainty of 0.10 (±0.05) and 0.05 (±0.025) respectively. On the other hand, our tool CARVE requires *less than one second* per instance to verify the robustness of the large-spread ensemble on all the instances of the test set, providing an exact estimate of the robustness of the model. As the maximum amount of verification time per instance increases, the number of instances on which SILVA is not able to verify the robustness of the model further decreases, e.g., 168 instances with a timeout of 120 seconds and 166 instances with a timeout of 180 seconds. Even though the robustness estimate of SILVA becomes more precise as the timeout per instance increases, i.e., the uncertainty on robustness decreases to 0.04 (±0.02) with a timeout per instance of 180 seconds, this process eventually hits a wall: the remaining 166 instances cannot be verified even when the timeout increases to 10 minutes per instance. Moreover, the improved precision comes at the cost of an higher total verification time: with a timeout of 120 seconds, SILVA requires in total 22,220 seconds to verify the traditional tree ensemble on the entire MNIST test set, while CARVE requires just 129 seconds in total, i.e., a reduction of two orders of magnitude. As expected, the results show the pitfalls of the complete robustness verification on traditional tree-ensembles and the improvements in the verification time enabled by the large-spread condition. Since the verification problem is NP-complete, there may be instances on which the verification time increases exponentially, while the large-spread condition allows one to train tree ensembles whose robustness can be verified in polynomial time on all the possible instances of the feature space.

5.3.2 Memory Efficiency. Our first experiment provides only a partial picture of the efficiency of the robustness verification and the reasons for the potential inefficiency of SILVA. Indeed, memory constraints should also be taken into account during robustness verification, since a high memory consumption may make the verification unfeasible on standard commercial systems.

In our second experiment, we compare the memory efficiency of SILVA and CARVE. In particular, we compare how many instances can be verified given a growing maximum memory consumption limit per instance, setting the maximum amount of verification time per instance to 10 minutes. The results of our experiment are shown in Figure 3b. The results highlight that SILVA may consume a lot of memory in order to provide precise robustness estimates. In the best scenario, with 100 GB of memory available, SILVA is still unable to verify the robustness of the model on 168 instances, providing just an approximate estimate of the robustness of the traditional tree-based ensemble with an uncertainty of 0.04 (±0.02). Even though the interval on which the robustness approximation is not so large in this setting, the plot shows that the number of instances that SILVA can not verify increases as the memory consumption limit decreases, expanding also the uncertainty of SILVA in the robustness estimation. For example, SILVA is not able to verify the robustness of the model on 216 and 342 instances with the memory consumption limit of 32 GB and 4 GB respectively, providing an uncertainty in the robustness estimates of 0.05 (±0.025) and



(a) Number of verified instances of the test set when varying the time limit in seconds for the verification.



(b) Number of verified instances of the test set when varying the maximum memory consumption limit for the verification.

Figure 3: Comparison of the time and memory efficiency of SILVA and CARVE on the MNIST dataset (we consider ensembles with 101 trees of maximum depth 6).

Table 5: Comparison of total verification time and maximum memory consumption of SILVA and CARVE on the MNIST test set. The last column reports the number of instances on which the verifier was not able to provide an answer because it exceeded the time or memory limits.

Tool	Total Time (s)	Memory (GB)	# Failures
SILVA	14,448	64	190
CARVE	129	0.03	0

0.08 (±0.04). Instead, CARVE manages to verify the robustness of the large-spread ensemble on all the MNIST test set using *less than 4GB of memory* per instance, providing an exact value of robustness. More precisely, the maximum memory consumption by CARVE is less than 1 GB in practice. The results confirm the efficiency in terms of memory consumption of our proposal and the unfeasibility of obtaining an exact value of robustness on traditional tree ensembles using a state-of-the-art verifier like SILVA when memory consumption constraints are imposed.

5.3.3 Efficiency Under Time and Memory Constraints. We finally perform a comparison between CARVE and SILVA when enforcing both a maximum verification time limit and a maximum memory consumption limit. In particular, we compare the total verification time, the maximum memory consumption and the number of instances on which the tool is not able to return an answer given a maximum verification time of 60 seconds per instance and a maximum memory consumption limit of 64 GB.

Table 5 shows the results of our experiment. The results confirm the observations from the previous sections: CARVE is far more efficient of SILVA in terms of both verification time and memory consumption. In particular, CARVE outperforms SILVA on the total verification time on the MNIST test set, verifying the large-spread ensemble on all the instances in just 129 seconds, thus being 112 times faster than SILVA (that requires 14,448 seconds). Moreover, the memory consumption of CARVE is more than 2,000 times lower than the one of SILVA, using just 0.03 GB of memory capacity, thus CARVE is usable on commodity hardware. Finally, SILVA is not able to provide an answer on 190 instances of the test set, providing an approximated robustness estimate with an uncertainty of 0.05 (±0.025), while CARVE is able to provide the exact robustness value. This provides clear evidence of the challenges of robustness verification for traditional tree ensembles: since robustness verification is NP-hard in general, even a state-of-the-art tool like SILVA is bound to fail on specific inputs. Our restriction to large-spread ensembles makes security verification feasible in polynomial time, thus ruling out such intractable cases which might occur in practical settings.

5.4 Efficiency of the Training Algorithm

Finally, we evaluate the time efficiency of the training algorithm for large-spread ensembles (Algorithm 3). Intuitively, the difficulty of enforcing the large-spread conditions depends on two factors: the model size and the adversarial perturbation k. Indeed, the larger is k, the higher becomes the distance to be enforced across thresholds in different trees. We then perform two experiments, each for different values of k: in the first, we fix the maximum tree depth at six and we vary the number of trees in $\{25, 51, 75, 101\}$; in the second, we fix the number of trees at 101 and we vary the maximum depth of the trees in $\{3, 4, 5, 6\}$. The presented times are measured for a specific hyper-parameter choice enabling successful training in all settings ($MAX\_ITER = 500$, $MULT = 6$, $INTV = [k, 1.5k]$, $l = 6$).

5.4.1 Number of Trees. Figure 4 shows the results of our first experiment. We observe that the time required for training a large-spread ensemble depends on the dataset, most likely because enforcing the large-spread condition might be easier or harder for different training data. When considering a number of trees less than or

equal to 75, the time required for training a large-spread ensemble is less than 150 seconds for all the considered datasets and adversarial perturbations. For example, the time required for training a large-spread ensemble of 75 trees is 28 seconds on MNIST and 145 seconds on Webspam when considering the largest adversarial perturbation. Similarly, training a large-spread ensemble is efficient when considering smaller adversarial perturbations: for the smallest perturbations, training time ranges from one second on the REWEMA dataset to 16 seconds on the Webspam dataset. This result is encouraging, because the trained models already obtain a reasonable accuracy on the test set and the range of adversarial perturbations might be small in practical cases.

On the downside, when considering larger models with 101 trees, the role of the adversarial perturbations on the training time becomes more significant. For example, training a large-spread ensemble with 101 trees under the largest adversarial perturbations required 137 seconds on MNIST and 1,835 seconds on Webspam. The motivation is that the cost of adding a tree to the ensemble increases as the size of the ensemble increases, because all the thresholds of the current ensemble must be adjusted with respect to the new tree. Fixing such violations to the large-spread condition is difficult for larger adversarial perturbations, because thresholds must be pushed farther away. This fact particularly affects the time required for training large-spread ensembles on the Webspam dataset: since some important features for the ensemble have a very skewed empirical distribution, the thresholds learned by the traditional tree-based ensembles for these features are close, thus separating them in an effective way is difficult and may require the training algorithm to perform many iterations.

5.4.2 Maximum Tree Depth. Figure 5 shows the results of our second experiment. We observe that training a large-spread ensemble of depth at most five requires at most 122 seconds for all the considered datasets and adversarial perturbations. For example, training a large-spread ensemble of 101 trees with maximum depth five takes 55 seconds on the Fashion-MNIST dataset and 122 seconds on the Webspam dataset. Moreover, the results confirm that training a large-spread ensemble considering small adversarial perturbations is efficient, e.g., the maximum time required for training a large-spread ensemble of 101 trees with maximum depth six, considering the smallest adversarial perturbation for each dataset, is 35 seconds.

However, we observe that, when considering large-spread ensembles with deeper trees, choosing a higher adversarial perturbation may determine a considerable increase in the time required for the training. The worst case is observed on the Webspam dataset, where the time required for training a large-spread ensemble of 101 trees with maximum depth six and $k = 0.0006$ is 1,835 seconds. Indeed, increasing the value of the depth of the trees in the ensemble causes an exponential growth in the number of nodes of the ensemble and enforcing the large-spread condition for higher perturbations is more difficult, thus more violations of the large-spread condition need to be fixed to add a single tree to the ensemble.

5.4.3 Discussion. Our experimental evaluation shows that the training algorithm for large-spread ensembles is efficient when the model size is relatively limited ($\leq$ 75 trees) or the adversarial perturbation is small. Concretely, the most challenging model including 75 trees could be trained in 145 seconds, while the most challenging model for the smallest adversarial perturbation could be trained in 35 seconds. When combining large model size with large adversarial perturbations, however, the training time can become higher. The worst case was observed on the Webspam dataset, where a model with 101 trees required 1,835 seconds to be trained under the largest adversarial perturbation. Nevertheless, this price is just paid for training: once the model is trained, robustness can be verified in polynomial time for thousands of instances. Also, such extreme cases only occurred on the Webspam dataset: for example, the most challenging models to train on Fashion-MNIST and REWEMA took just 113 seconds and 13 seconds respectively. We find these results appropriate for our first evaluation of large-spread ensembles, in particular because our implementation of LSE is not heavily optimized, and we plan to design more efficient training algorithms for large-spread ensembles as future work.

5.5 Take-Away Messages

Our experimental evaluation shows that:

- Large-spread ensembles sacrifice some predictive power with respect to traditional tree ensembles, yet their accuracy remains way higher than the majority class of the test set. Even better, in several cases the accuracy of large-spread ensembles is equal to the accuracy of traditional tree ensembles.
- Large-spread ensembles are generally more robust than traditional tree ensembles. This empirical observation is a useful byproduct of the large-spread condition, which makes it harder to craft evasion attacks which are effective against multiple trees in the ensemble.
- Our verification tool for large-spread ensemble CARVE is much more efficient than SILVA, a state-of-the-art verifier for traditional tree ensembles. Improvements are due to both verification time and memory consumption.
- Our training tool LSE is efficient for training large-spread ensembles of moderate size or when considering small adversarial perturbations. The time required for training a large-spread ensemble may increase when the model is large and the considered perturbation is high for the chosen dataset.

Moreover, we showed that SILVA can provide just approximate robustness estimates in some experimental settings, even when provided with extremely high time and memory bounds (10 minutes per instance, 100 GB of RAM). Conversely, CARVE can compute the exact value of robustness using just limited time and memory (1 second per instance, 1 GB of RAM). This shows the effectiveness of the verifiable learning paradigm: models trained with formal verification in mind can be verified in a matter of seconds even on traditional commercial hardware, contrary to traditional machine learning models which cannot be accurately verified even when extremely powerful servers are available.

6 RELATED WORK

We already mentioned that prior work studied the complexity of the robustness verification problem for decision tree ensembles [24, 42]. This problem was proved to be NP-complete for arbitrary L_p-norm attackers, even when restricting the model shape to decision stump ensembles [1, 42]. To the best of our knowledge, we are the first

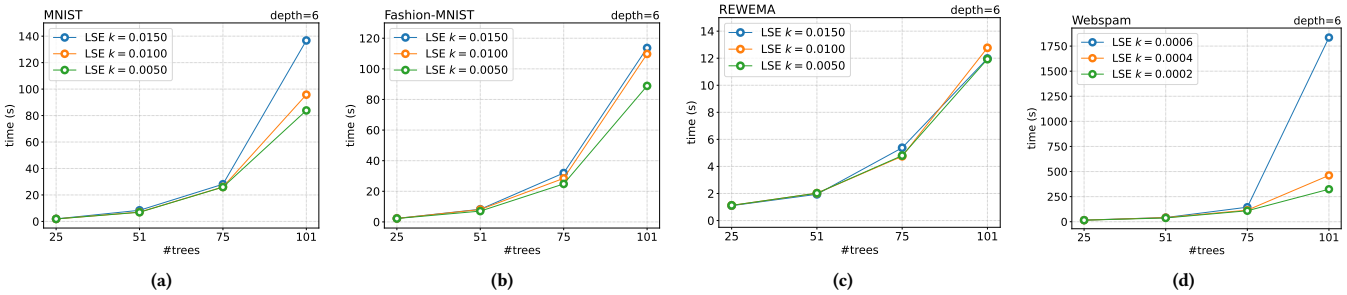


Figure 4: Efficiency of LSE when varying the number of trees of the large-spread ensemble.

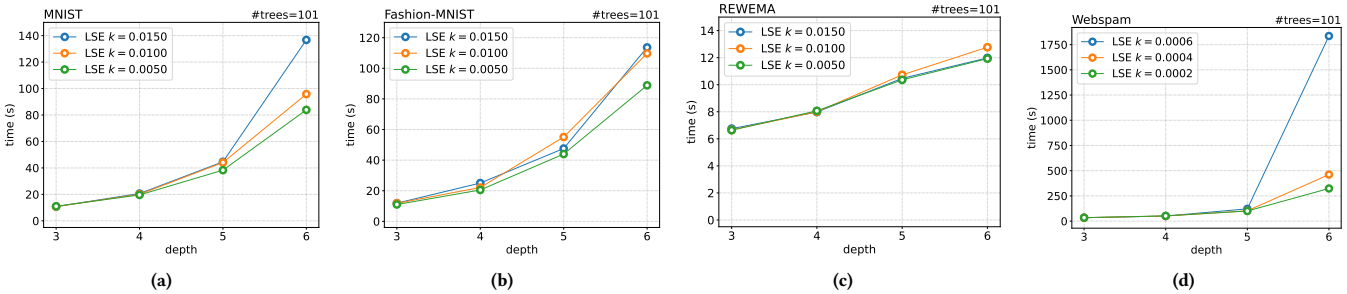


Figure 5: Efficiency of LSE when varying the maximum depth of the trees of the large-spread ensemble.

to identify a specific class of decision tree ensembles enabling robustness verification in polynomial time. Prior work on robustness verification for decision tree ensembles proposed different techniques, such as exploiting equivalence classes extracted from the tree ensemble [37], integer linear programming [24], a reduction to the max clique problem [13], abstract interpretation [8, 32] and satisfiability modulo theory (SMT) solving [17, 19, 34]. Though effective in many cases, these techniques still have to deal with the exponential complexity of the robustness verification problem, so they are bound to fail for large ensembles and complex datasets. We experimentally showed that a state-of-the-art verifier like SILVA [32] is much less efficient than our verifiable learning approach, supporting verification in polynomial time, and can only compute approximate robustness estimates in practical cases. Moreover, our LSE training algorithm produces tree ensembles that are in general more robust than the traditional counterparts as a side-effect of imposing that the thresholds of different trees are sufficiently far away. Several papers in the literature discussed new algorithms for training tree ensembles that are robust to evasion attacks [1, 9–12, 14, 21, 24, 33, 39–41], but our work is complementary to them. Indeed, our primary goal is not enforcing robustness, which is a byproduct of our training algorithm, but supporting efficient robustness verification of the trained models. We also acknowledge that our work solely focuses on the classic definition of robustness, known as *local* robustness in more recent literature discussing global robustness and related properties [6, 15, 28]. This line of research aims to achieve security verification independently of the choice of a specific test set, enhancing the credibility of security proofs. Given that local robustness remains popular and is easier to deal with, we stick to it in this paper and we leave the extension of our framework to global robustness as future work. We think that this would be feasible, as our large-spread condition represents a data-independent structural property of tree ensembles.

It is worth mentioning that a lot of work has been done on the robustness verification of deep neural networks (DNNs). Classic approaches for exact verification often do not scale to large DNNs, as for tree ensembles, and they are typically based on SMT [22, 25, 26] and integer linear programming [2, 18, 29, 36]. To mitigate the scalability problems of robustness verification, different proposals have been done, such as shrinking the original DNN through pruning [20] and finding specific classes of DNNs that empirically enable more efficient robustness verification [23]. Xiao et. al. [43] proposed the idea of *co-designing* model training and verification, i.e., training models that show reasonable accuracy and robustness, while better enabling exact verification. In particular, their work proposes a training algorithm for DNNs that encourages weight sparsity and ReLU stability, two properties that improve the efficiency of verification through SMT solving. There are significant differences between these lines of work and ours. First, prior techniques only provide empirical efficiency guarantees, while our proposal leads to a formal complexity reduction of the robustness verification problem through the design of a polynomial time algorithm. Moreover, our research deals with tree ensembles rather than DNNs.

Finally, we observe that recent work explored the adversarial robustness of model ensembles [44]. The main result of this work proved that the combination of "diversified gradient" and "large

confidence margin" are sufficient and necessary conditions for certifiably robust ensemble models. While this result cannot be directly applied to non-differentiable models such as decision tree ensembles, the intuition of diversifying models is similarly captured by our large-spread condition. We plan to explore any intriguing connections with this proposal as future work.

7 CONCLUSION

We introduced the general idea of *verifiable learning*, i.e., the adoption of training algorithms designed to learn restricted model classes amenable for efficient security verification. We applied this idea to decision tree ensembles, identifying the class of *large-spread* ensembles. We showed that this class of ensembles admits robustness verification in polynomial time, whereas the problem is NP-hard for general decision tree models. We then proposed a pruning-based training algorithm to learn large-spread ensembles from traditional decision tree ensembles. Our experiments on public datasets show that large-spread ensembles sacrifice a limited amount of the predictive power of traditional tree ensembles, but their robustness is normally higher and much more efficient to verify. This makes large-spread ensembles appealing in the adversarial setting.

As future work, we plan to investigate the use of verifiable learning also for other popular model classes, e.g., neural networks. Moreover, we want to explore different training algorithms for large-spread ensembles and compare their effectiveness against the pruning-based approach proposed in this paper.

Acknowledgements. We thank the reviewers for their constructive feedback, which has greatly contributed to the improvement of this paper. This research was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU and by project EFRA (101093026) under the Horizon Europe programme.

REFERENCES

[1] Maksym Andriushchenko and Matthias Hein. 2019. Provably robust boosted decision stumps and trees against adversarial attacks. In *NeurIPS*.
[2] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya V. Nori, and Antonio Criminisi. 2016. Measuring Neural Net Robustness with Constraints. In *NeurIPS*.
[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *ECML PKDD*.
[4] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (2001), 5–32.
[5] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth.
[6] Stefano Calzavara, Lorenzo Cazzaro, Claudio Lucchese, Federico Marcuzzi, and Salvatore Orlando. 2022. Beyond robustness: Resilience verification of tree-based classifiers. *Comput. Secur.* 121 (2022).
[7] Stefano Calzavara, Lorenzo Cazzaro, Giulio Ermanno Pibiri, and Nicola Prezza. 2023. Verifiable Learning for Robust Tree Ensembles. *CoRR* abs/2305.03626 (2023). https://doi.org/10.48550/arXiv.2305.03626
[8] Stefano Calzavara, Pietro Ferrara, and Claudio Lucchese. 2020. Certifying Decision Trees Against Evasion Attacks by Program Analysis. In *ESORICS*.
[9] Stefano Calzavara, Claudio Lucchese, Federico Marcuzzi, and Salvatore Orlando. 2021. Feature partitioning for robust tree ensembles and their certification in adversarial scenarios. *EURASIP J. Inf. Secur.* 2021, 1 (2021), 12.
[10] Stefano Calzavara, Claudio Lucchese, and Gabriele Tolomei. 2019. Adversarial Training of Gradient-Boosted Decision Trees. In *CIKM*.
[11] Stefano Calzavara, Claudio Lucchese, Gabriele Tolomei, Seyum Assefa Abebe, and Salvatore Orlando. 2020. Treant: training evasion-aware decision trees. *Data Min. Knowl. Discov.* 34, 5 (2020), 1390–1420.
[12] Hongge Chen, Huan Zhang, Duane S. Boning, and Cho-Jui Hsieh. 2019. Robust Decision Trees Against Adversarial Examples. In *ICML*.
[13] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane S. Boning, and Cho-Jui Hsieh. 2019. Robustness Verification of Tree-based Models. In *NeurIPS*.
[14] Yizheng Chen, Shiqi Wang, Weifan Jiang, Asaf Cidon, and Suman Jana. 2021. Cost-Aware Robust Tree Ensembles for Security Applications. In *USENIX Security Symposium*.
[15] Yizheng Chen, Shiqi Wang, Yue Qin, Xiaojing Liao, Suman Jana, and David A. Wagner. 2021. Learning Security Classifiers with Verified Global Robustness Properties. In *ACM CCS*.
[16] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. *ACM Trans. Priv. Secur.* 24, 4 (2021), 27:1–27:31.
[17] Laurens Devos, Wannes Meert, and Jesse Davis. 2021. Verifying Tree Ensembles by Reasoning about Potential Instances. In *SDM*.
[18] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NFM*.
[19] Gil Einziger, Maayan Goldstein, Yaniv Sa'ar, and Itai Segall. 2019. Verifying Robustness of Gradient Boosted Models. In *AAAI*.
[20] Dario Guidotti, Francesco Leofante, Luca Pulina, and Armando Tacchella. 2020. Verification of Neural Networks: Enhancing Scalability Through Pruning. In *ECAI*.
[21] Jun-Qi Guo, Ming-Zhuo Teng, Wei Gao, and Zhi-Hua Zhou. 2022. Fast Provably Robust Decision Trees and Boosting. In *ICML*.
[22] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety Verification of Deep Neural Networks. In *CAV*.
[23] Kai Jia and Martin C. Rinard. 2020. Efficient Exact Verification of Binarized Neural Networks. In *NeurIPS*.
[24] Alex Kantchelian, J. D. Tygar, and Anthony D. Joseph. 2016. Evasion and Hardening of Tree Ensemble Classifiers. In *ICML*.
[25] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *CAV*.
[26] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljic, David L. Dill, Mykel J. Kochenderfer, and Clark W. Barrett. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV*.
[27] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *NeurIPS*.
[28] Klas Leino, Zifan Wang, and Matt Fredrikson. 2021. Globally-Robust Neural Networks. In *ICML*.
[29] Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR* abs/1706.07351 (2017).
[30] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *ICLR*.
[31] Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin T. Vechev. 2023. Certified Training: Small Boxes are All You Need. In *ICLR*.
[32] Francesco Ranzato and Marco Zanella. 2020. Abstract Interpretation of Decision Tree Ensemble Classifiers. In *AAAI*.
[33] Francesco Ranzato and Marco Zanella. 2021. Genetic adversarial training of decision trees. In *GECCO*.
[34] Naoto Sato, Hironobu Kuruma, Yuichiroh Nakagawa, and Hideto Ogawa. 2020. Formal Verification of a Decision-Tree Ensemble Model and Detection of Its Violation Ranges. *IEICE Trans. Inf. Syst.* 103-D, 2 (2020), 363–378.
[35] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.
[36] Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *ICLR*.
[37] John Törnblom and Simin Nadjm-Tehrani. 2020. Formal verification of input-output mappings of tree ensembles. *Sci. Comput. Program.* 194 (2020), 102450.
[38] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness May Be at Odds with Accuracy. In *ICLR*.
[39] Daniël Vos and Sicco Verwer. 2021. Efficient Training of Robust Decision Trees Against Adversarial Examples. In *ICML*.
[40] Daniël Vos and Sicco Verwer. 2022. Adversarially Robust Decision Tree Relabeling. In *ECML PKDD*.
[41] Daniël Vos and Sicco Verwer. 2022. Robust Optimal Classification Trees against Adversarial Examples. In *AAAI*.
[42] Yihan Wang, Huan Zhang, Hongge Chen, Duane S. Boning, and Cho-Jui Hsieh. 2020. On Lp-norm Robustness of Ensemble Decision Stumps and Trees. In *ICML*.
[43] Kai Yuanqing Xiao, Vincent Tjeng, Nur Muhammad (Mahi) Shafiullah, and Aleksander Madry. 2019. Training for Faster Adversarial Robustness Verification via Inducing ReLU Stability. In *ICLR*.
[44] Zhuolin Yang, Linyi Li, Xiaojun Xu, Bhavya Kailkhura, Tao Xie, and Bo Li. 2022. On the Certified Robustness for Ensemble Models and Beyond. In *ICLR*.