# Verifying Robustness of Gradient Boosted Models

**Gil Einziger, Maayan Goldstein, Yaniv Sa'ar, Itai Segall**
Nokia, Bell Labs
gilein@bgu.ac.il, {maayan.goldstein, yaniv.saar, itai.segall}@nokia-bell-labs.com

## Abstract

Gradient boosted models are a fundamental machine learning technique. Robustness to small perturbations of the input is an important quality measure for machine learning models, but the literature lacks a method to prove the robustness of gradient boosted models.

This work introduces VERIGB, a tool for quantifying the robustness of gradient boosted models. VERIGB encodes the model and the robustness property as an SMT formula, which enables state of the art verification tools to prove the model's robustness. We extensively evaluate VERIGB on publicly available datasets and demonstrate a capability for verifying large models. Finally, we show that some model configurations tend to be inherently more robust than others.

## 1 Introduction

Gradient boosted models are fundamental in machine learning and are among the most popular techniques in practice. They are known to achieve good accuracy with relatively small models, and are attractive in numerous domains ranging from computer vision to transportation (Viola and Jones 2001; Yang et al. 2015; Caruana and Niculescu-Mizil 2006; Freund and Schapire 1997; Chapelle and Chang 2010; Zhang and Haghani 2015). They are easy to use as they do not require normalization of input features, and they support custom loss functions as well as classification and regression. Finally, the method has a solid theoretical grounding (Mason et al. 1999).

Machine learning models are often vulnerable to adversarial perturbations, which may cause catastrophic failures (e.g., by misclassification of a traffic sign). Specifically, Figure 1 exemplifies that gradient boosted models are indeed vulnerable to such perturbations. Thus, identifying which models are robust to such manipulations and which are not is critical. Indeed, numerous works suggested training techniques that increase the robustness (Leistner et al. 2009; Sun, Todorovic, and Li 2007). However, there is currently no method to formally verify gradient boosted models. Furthermore, it is not clear how the configuration parameters of such models affect their robustness. These knowledge gaps make it challenging to guarantee the reliability of gradient boosted solutions.



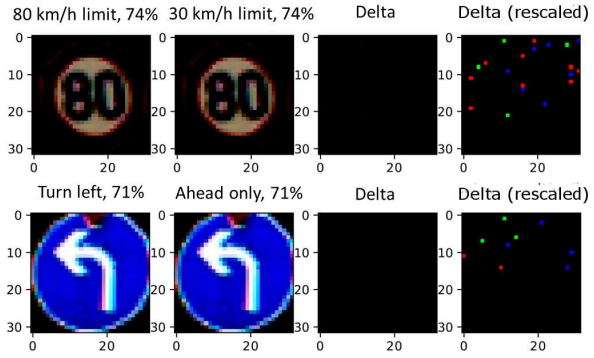


Figure 1: Example of the lack of robustness in a gradient boosted model trained over a traffic signs dataset. In the first row, an "80 km/h speed limit" sign is misclassified as a "30 km/h speed limit". In the second row, a "turn left" sign is misclassified as "ahead only". Observe in the third column (delta, computed as the difference in pixel values of the two images) that the applied changes are barely visible to the naked eye (delta of +/-3 in the range of 256 values per pixel per color). The fourth column highlights the modified pixels.

In the last couple of decades, formal methods successfully increased the reliability of numerous software and hardware systems. Such success gave rise to diverse verification methods such as model checking, termination analysis, and abstract interpretation. Formal methods are especially appealing in situations where the cost of mistakes is exceptionally high. Examples include mission-critical solutions as well as mass-produced hardware. Unfortunately, machine learning models are fundamentally different from traditional software artifacts, and we cannot directly use existing verification techniques for machine learning models. The research community already started addressing the problem for neural network models (Pulina and Tacchella 2010; Katz et al. 2017; Huang et al. 2017; Gehr et al. 2018; Narodytska et al. 2018). Here we focus on an area that has not been covered so far – verification of robustness of gradient boosted models.

The main contribution of this work is the VERIGB tool for verifying the robustness of gradient boosted models.

VERIGB encapsulates novel and formally proven methods that translate such models, and robustness properties into SMT formulas. Then, we feed these formulas to a standard SMT solver, which proves the robustness or provides a counter-example. VERIGB includes runtime optimizations that make the verification process practical. We extensively evaluate it with public datasets and demonstrate scalability for large and accurate models. Finally, we highlight that some model configurations are fundamentally more robust than others.

The rest of this paper is organized as follows: In Section 2 we provide background on logic, decision trees, and gradient boosted models. Next, in Section 3, we formally define the robustness properties. The SMT formula representation of gradient boosted models is given in Section 4, and that of the robustness property in Section 5. Next, Section 6 suggests optimizations of these encodings improving their runtime. Section 7 evaluates VERIGB on several publicly-available datasets, while Section 8 surveys related work. We conclude in Section 9, which discusses the implications of our work and suggests directions for future research.

# 2 Preliminaries

## 2.1 Logic and Linear Arithmetic

A *propositional formula* is defined inductively as one of the following: *(i)* 'True' and 'False' constants (T and F). *(ii)* a variable $x_i \in \{x_1, \ldots, x_m\}$; *(iii)* if $\varphi$ and $\psi$ are propositional formulas then so are $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \rightarrow \psi$, $\varphi \leftrightarrow \psi$ (with their usual interpretation). Given a propositional formula $\varphi$, the Boolean satisfiability problem (SAT) determines whether there exists an assignment under which $\varphi$ evaluates to True.

*Satisfiability Modulo Theories (*SMT*)* extends the Boolean SAT problem by combining a variety of underlying theories (Barrett et al. 2009). We use the linear real arithmetic theory, which extends the propositional fragment with all rational number constants, and with the symbols: $\{+, -, \cdot, \leq, \geq\}$. A formula $\varphi$ (be that an SMT or SAT instance) is said to be *satisfiable*, if $\varphi$ evaluates to True for some assignment $\vec{x} \in \mathbb{R}^m$. If there is no such assignment, we say that $\varphi$ is *unsatisfiable*.

## 2.2 Decision Trees

Decision trees are functions that receive an assignment $\vec{x} \in \mathbb{R}^m$ and return a value. Formally, a *decision tree structure* (DTS) $D = \langle N, I, L\rangle$ is defined as follows:

- $N = \{n_1, \ldots, n_k\}$: is the set of nodes in the tree, and $n_1$ is defined to be the *root node* of the tree.
- $I \subseteq N$: is the subset of internal nodes in the tree. An *internal node* is a triplet $n = \langle \mathsf{S}_n, \mathsf{T}_n, \mathsf{F}_n\rangle$, where $\mathsf{S}_n$ is a condition expressing the decision of node $n$ (an SMT formula), and $\mathsf{T}_n \in N$ (resp., $\mathsf{F}_n \in N$) is the target successor node when the condition evaluates to True (resp., False).
- $L = N \backslash I$: is the subset of leaf nodes in the tree, i.e. nodes for which there is no successor. A *leaf node* $n = \langle \mathsf{W}_n\rangle$ also has a weight $\mathsf{W}_n \in \mathbb{R}$.

Intuitively, S (resp., T and F) is a dictionary that associates to every $n \in I$ a condition $\mathsf{S}_n$ (resp., a positive child $\mathsf{T}_n \in N$ and a negative child $\mathsf{F}_n \in N$). W is a dictionary that associates to every $n \in L$ a weight $\mathsf{W}_n \in \mathbb{R}$.

A DTS $D$ is said to be *well-formed* if, and only if, every node $n \in N$ has exactly one predecessor node, except for the root node that has no predecessor. In a well-formed tree, we denote by $\mathsf{P}_n$ the predecessor of node $n \in N$. Given an input vector $\vec{x} \in \mathbb{R}^m$, the *valuation of a* DTS $D$ *on* $\vec{x}$ is a function $\hat{D} : \mathbb{R}^m \rightarrow \mathbb{R}$. Tree $D$ is traversed according to $\vec{x}$, ending in a leaf node $n \in L$, and function $\hat{D}(\vec{x})$ is the weight of that node, i.e. $\mathsf{W}_n \in \mathbb{R}$.

## 2.3 Gradient Boosted Trees

Gradient boosted regression is an ensemble technique that constructs a strong learner by iteratively adding weak learners (typically decision trees) (Mason et al. 1999). Formally, a *Gradient Boosted Regressor (*GBR*)* is a sequence of $r$ decision trees $R = \langle D_1, \ldots, D_r\rangle$. Given an input vector $\vec{x} \in \mathbb{R}^m$, the *valuation of a* GBR $R$ is the sum of valuations of its $r$ decision trees. That is, $\hat{R}(\vec{x}) = \sum_{i=1}^{r} \hat{D}_i(\vec{x})$.

Gradient boosted classification is a tree ensemble technique that constructs a strong learner per each class (again, by iteratively adding weak learners), to assign a class for a given input. Let $c$ be the number of classes. Formally, a *Gradient Boosted Classifier* (GBC) $C = \langle R_1, \ldots, R_c\rangle$ is a sequence of $c$ gradient boosted regressors, where regressor $R_j = \langle D_1^j, \ldots, D_r^j\rangle$. Given an input vector $\vec{x} \in \mathbb{R}^m$, the *valuation of* $C$, valuates all $c$ regressors over $\vec{x}$ and returns the class associated with the maximal value, namely: $\hat{C}(\vec{x}) = \arg\max_j(\hat{R}_j(\vec{x}))$. We assume that there is an association between each input vector and a single class[1].

# 3 Robustness of Machine Learning Models

Robustness means that small perturbations in the input have little effect on the outcome. That is, for classifiers the classification remains the same, and for regressors, the change in valuation is bounded. This section formally defines robustness properties, in a similar manner to (Pulina and Tacchella 2012; Narodytska et al. 2018; Katz et al. 2017; Moosavi-Dezfooli et al. 2017).

Consider a regression model $R$, and let $\hat{R}(\vec{x})$ be the valuation function of $R$ for an input $\vec{x} \in \mathbb{R}^m$. We define *local adversarial* $(\epsilon, \delta)$*-robustness for an input* $\vec{x}$, as follows:

**Definition 3.1** (local adversarial robustness of regressors)**.** *A regression model $R$ is said to be $(\epsilon, \delta)$-robust for an input $\vec{x}$, if for every input $\vec{x}'$ such that $||\vec{x} - \vec{x}'||_p < \epsilon$, the output is bound by $\delta$, i.e., $|\hat{R}(\vec{x}) - \hat{R}(\vec{x}')| \leq \delta$.*

Here, $||\vec{x} - \vec{x}'||_p$ is used to specify the distance between two vectors $\vec{x}$ and $\vec{x}'$ according to some norm $p$. For example, one may compute the distance between two images as the maximal difference between pairs of corresponding pixels (i.e., $p = \infty$), or the sum of these differences (i.e., $p = 1$).

[1] In cases where multiple regressors return the same maximal value we can break the symmetry using their indices.

Throughout this paper we use norm $p = \infty$, but our techniques are applicable to any norm that is linear to the input.

Next, consider a classification model $C$ and let $\hat{C}(\vec{x})$ be the valuation function of $C$ for an input $\vec{x} \in \mathbb{R}^m$. We define *local adversarial $\epsilon$-robustness for an input $\vec{x}$* as follows:

**Definition 3.2** (local adversarial robustness of classifiers)**.** *A classification model $C$ is said to be $\epsilon$-robust for an input $\vec{x}$, if for every input $\vec{x}'$ such that $||\vec{x} - \vec{x}'||_p < \epsilon$, the output does not change its classification, i.e., $\hat{C}(\vec{x}) = \hat{C}(\vec{x}')$.*

The above definitions aim to certify a given input but do not guarantee much regarding the model itself. Therefore, we extend these definitions to capture the behavior over a set of inputs $A$. We define *$\rho$-universal adversarial $(\epsilon, \delta)$-robustness on a set of inputs $A$*, as follows:

**Definition 3.3** (universal adversarial robustness of regressors)**.** *A regression model $R$ is said to be $\rho$-universally $(\epsilon, \delta)$-robust over the set of inputs $A$, if it is $(\epsilon, \delta)$-robust for at least $\rho \cdot |A|$ inputs in $A$.*

Finally, we extend the classifier definition of local $\epsilon$-robustness, and define *$\rho$-universal adversarial $\epsilon$-robustness on a set of inputs $A$*, as follows:

**Definition 3.4** (universal adversarial robustness of classifiers)**.** *A classification model $C$ is said to be $\rho$-universally $\epsilon$-robust over the set of inputs $A$, if it is $\epsilon$-robust for at least $\rho \cdot |A|$ inputs in $A$.*

Definition 3.3 and Definition 3.4 capture the universal adversarial robustness properties for regressors and classifiers. The parameter $\epsilon$ determines the allowed perturbation change, that is, how much an attacker can change the input. For regressors, we also require the parameter $\delta$ that defines the acceptable change in the output, while for classifiers we require that the classification stays the same. Finally, the parameter $\rho$ measures the portion of robust inputs. In Section 7, we evaluate the $\rho$ values of varying models instead of selecting a $\rho$ value in advance.

# 4 Encodings of Gradient Boosted Models

This section explains the encoding of gradient boosted models into SMT formulas. We start by translating a single path in a decision tree and then work our way up until we end up with a formula for the entire model.

## 4.1 Encoding of Decision Trees

Given a well-formed DTS $D = \langle N, I, L \rangle$ and a leaf $l \in L$, we define $path(l)$ to be the set of nodes on the path in the tree between the leaf node $l$ and the root node $n_1$ (including both nodes). We define the *encoding of leaf $l$ in tree $D$* to be the formula $\pi(l)$ as follows:

$$\pi(l): \bigwedge_{n \in path(l) \setminus \{n_1\}} \begin{pmatrix} \mathsf{T}_{\mathsf{P}_n} = n \rightarrow \ \mathsf{S}_{\mathsf{P}_n} \wedge \\ \mathsf{F}_{\mathsf{P}_n} = n \rightarrow \neg \mathsf{S}_{\mathsf{P}_n} \end{pmatrix} \wedge (wl = \mathsf{W}_l)$$

The encoding $\pi(l)$ restricts the *decision tree valuation variable* $wl$ to be the weight of the leaf ($wl = \mathsf{W}_l$), and for each node $n$ in the path except for the root, if node $n$ is the positive child of its parent ($\mathsf{T}_{\mathsf{P}_n} = n$) then the parent condition should hold ($\mathsf{S}_{\mathsf{P}_n}$), and if node $n$ is the negative child of its parent ($\mathsf{F}_{\mathsf{P}_n} = n$) then the negation of the parent condition should hold ($\neg \mathsf{S}_{\mathsf{P}_n}$).

**Lemma 4.1** (leaf encoding)**.** *Let $\hat{D}$ be the valuation function of the well-formed tree $D$. If $\pi(l)$ evaluates to* True*, then there exists a truth assignment $\vec{x} \in \mathbb{R}^m$, $wl \in \mathbb{R}$ such that $\hat{D}(\vec{x})$ reaches leaf $l$ , and $\hat{D}(\vec{x}) = \mathsf{W}_l = wl$.*

*Proof.* Assume that the leaf encoding $\pi(l)$ evaluates to True, then there exists a truth assignment $\vec{x} \in \mathbb{R}^m$, $wl \in \mathbb{R}$. Since the tree is well-formed and following the definition of $path(l)$, we know that every internal node $n' \in path(l) \cap I$ is a predecessor of some node $n \in path(l)$, i.e., $n' = \mathsf{P}_n$. If $n$ is the positive successor of $n'$, then $(\mathsf{T}_{\mathsf{P}_n} = n)$ holds, implying that $\mathsf{S}_{n'}$ holds for $\vec{x}$ as well. Thus, when the valuation of $\hat{D}(\vec{x})$ traverses tree $D$ and reaches node $n'$, we know that it indeed turns to the positive child. The same reasoning applies to the negative successor of $n'$. By applying this reasoning recursively from the root node, we show that the traversal of the valuation reaches leaf $l$, and outputs $\hat{D}(\vec{x}) = \mathsf{W}_l = wl$. □

Given DTS $D = \langle N, I, L \rangle$, we now define the *encoding of tree $D$* to be the formula $\Pi(D)$ as follows:

$$\Pi(D): \quad \bigvee_{l \in L} \pi(l)$$

Namely, $\Pi(D)$ is a disjunction of formulas, where each disjunct represents a concrete path to one of the leaves in $D$ and its respective valuation.

**Lemma 4.2** (tree encoding)**.** *Let $\hat{D}$ be the valuation function of the well-formed tree $D$. If $\Pi(D)$ evaluates to* True*, then there exists a truth assignment $\vec{x} \in \mathbb{R}^m$, $wl \in \mathbb{R}$, and a single leaf $l \in L$ for which $\hat{D}(\vec{x})$ reaches $l$ and outputs $\hat{D}(\vec{x}) = \mathsf{W}_l = wl$.*

*Proof.* Assume that the tree encoding $\Pi(D)$ evaluates to True, then there exists a truth assignment $\vec{x} \in \mathbb{R}^m$, $wl \in \mathbb{R}$. Clearly, at least one clause in $\Pi(D)$ evaluates to True. Since tree $D$ is well formed, at most one clause in $\Pi(D)$ evaluates to True, otherwise there exists an internal node in the path $n \in path(l) \cap I$ for which $\mathsf{S}_n$ is inconsistent over $\vec{x}$. Therefore, there exists exactly one clause in $\Pi(D)$ that evaluates to True, and exactly one leaf $l \in L$ for which $\pi(l)$ evaluates to True. If $\pi(l)$ evaluates to True, then following the same reasoning of Lemma 4.1, the truth assignment $\vec{x} \in \mathbb{R}^m$, $wl \in \mathbb{R}$ reaches leaf $l$ and outputs $\hat{D}(\vec{x}) = \mathsf{W}_l = wl$. □

## 4.2 Encoding of Gradient Boosted Trees

Given GBR $R = \langle D_1, \ldots, D_r \rangle$ and following Lemma 4.1, and Lemma 4.2, we define the *encoding of regressor $R$* to be the formula $\Upsilon(R)$ as follows:

$$\Upsilon(R): \quad \Big( \bigwedge_{i=1}^{r} \Pi(D_i) \Big) \wedge out = \sum_{i=1}^{r} wl_i$$

Intuitively, $\Upsilon(R)$ consists of two parts: *(i)* the conjunction of all tree encodings, ensuring that the decision tree valuation variables of each tree $wl_1, \ldots, wl_r$ are restricted to their

respective tree valuations; and *(ii)* a restriction of the *regressor valuation variable* $out$ to be the sum of all decision tree valuation variables $wl_1, \ldots, wl_r$. Therefore, encoding $\Upsilon(R)$ characterizes regressor $R$.

**Theorem 4.3** (regressor encoding)**.** *Let $\hat{R}$ be the valuation function of regressor $R$. If $\Upsilon(R)$ evaluates to* True*, then there exist a truth assignment $\vec{x} \in \mathbb{R}^m$, $out \in \mathbb{R}$, such that $\hat{R}(\vec{x}) = out$.*

*Proof.* The proof follows from the definitions and Lemma 4.2. □

Given GBC $C = \langle R_1, \ldots, R_c \rangle$ and following Theorem 4.3, we define the *encoding of classifier* $C$ to be the formula $\Gamma(C)$ as follows:

$$\Gamma(C): \quad \bigwedge_{j=1}^{c} \Upsilon(R_j) \wedge \bigvee_{j=1}^{c} \left( arg = j \leftrightarrow \bigwedge_{k=1}^{c} out_j > out_k \right)$$

Intuitively, $\Gamma(C)$ consists of two parts: *(i)* the conjunction of all regressor encodings, ensuring that the regressor valuation variables $out_1, \ldots, out_r$ are restricted to their respective regressor valuations; and *(ii)* a restriction of the *classifier valuation variable* $arg$ to be the maximal regressor valuation (i.e., operator $\arg\max$). Therefore, $\Gamma(C)$ charactarizes classifier $C$.

**Theorem 4.4** (classifier encoding)**.** *Let $\hat{C}$ be the valuation function of classifier $C$. If $\Gamma(C)$ evaluates to* True*, then there exist a truth assignment $\vec{x} \in \mathbb{R}^m$, $arg \in \{1, \ldots, c\}$, such that $\hat{C}(\vec{x}) = arg$.*

*Proof.* The proof follows from the definitions, theorem, and lemmas above. □

# 5 Encodings of Local Robustness Properties

In this section, we encode the local robustness properties defined in Section 3. Recall that a regression model (resp., classification model) satisifies local adversarial robustness for an input $\vec{x}$ (Definitions 3.1, and 3.2), if for all $\vec{x}'$, if $||\vec{x} - \vec{x}'||_p < \epsilon$, then the difference between the valuation of $\vec{x}$, and that of $\vec{x}'$ is bound (resp., we get the same classification for $\vec{x}$, and for $\vec{x}'$).

Our goal is to find whether there exists an assignment to $\vec{x}'$ that satisfies both the model encoding, and the negation of the local adversarial robustness property. An assignment $\vec{x}'$ that satisfies both conjuncts constitutes a *counter-example* that disproves local adversarial robustness of the given input $\vec{x}$. Alternatively, local adversarial robustness holds if there is no such assignment.

Given an input $\vec{x}$, and $\epsilon, \delta \geq 0$, we define the *encoding of local adversarial robustness* to be a formula $\Phi$ as follows:

$$\Phi: \quad \phi \wedge \bigwedge_{i=1}^{m} \begin{cases} |x_i - x_i'| \leq \epsilon, & x_i \in \mathbb{R} \\ x_i' \in \{v \in \mathbb{N} \,:\, |x_i - v| \leq \epsilon\}, & x_i \in \mathbb{N} \end{cases}$$

Where $\phi$ is $|\hat{R}(\vec{x}) - \hat{R}(\vec{x}')| \geq \delta$ for regression model, and $\phi$ is $\hat{C}(\vec{x}) \neq \hat{C}(\vec{x}')$ for classification model. Note that the second range of conjuncts in the expression, characterizes the allowed pertubations ($||\vec{x} - \vec{x}'||_p < \epsilon$) for norm $p = \infty$, which is handled differently for real, and integer features.

# 6 Optimizations

While the construction in Sections 4 and 5 is sound and complete, it is not always the most efficient one. Thus, we now provide two optimizations based on eliminating redundant clauses that cannot be satisfied, and on parallelizing the verification process.

## 6.1 Pruning

"Pruning" is a somewhat overloaded term. In the context of machine learning, pruning typically refers to the process of removing sections of decision trees that have little impact on the outcome, thus reducing over-fitting. In the model-checking community, pruning is the process of trimming unreachable parts of the search space, thus helping the model checker focus its search.

Our approach combines these two notions. Namely, we remove all unsatisfiable leaf clauses with respect to the robustness parameter ($\epsilon$), which allows for faster calculation. Formally, given DTS $D = \langle N, I, L \rangle$ and property $\Phi$, we define the $\Phi$*-pruned encoding of leaf* $l$ *in tree* $D$ to be:

$$\pi^{\Phi}(l) = \begin{cases} \pi(l), & \pi(l) \wedge \Phi \text{ is satisfiable} \\ \mathsf{False}, & \pi(l) \wedge \Phi \text{ is unsatisfiable} \end{cases}$$

Note that pruning can be applied to diverse properties, but this work is focused on the robustness property.

Next, we define the corresponding $\Upsilon^{\Phi}$ (resp., $\Gamma^{\Phi}$) to be the $\Phi$*-pruned encoding of regressor* $R$ (resp., $\Phi$*-pruned encoding of classifier* $C$), which replaces each occurrence of leaf encoding $\pi(l)$ with its pruned version $\pi^{\Phi}(l)$. The following theorem establishes the correctness of $\Phi$-pruning:

**Theorem 6.1** (safe pruning)**.**

1. ***Regressor:*** *the conjunction $\Upsilon(R) \wedge \Phi$ is satisfiable, if and only if, the conjunction $\Upsilon^{\Phi}(R) \wedge \Phi$ is satisfiable.*
2. ***Classifier:*** *the conjunction $\Gamma(C) \wedge \Phi$ is satisfiable, if and only if, the conjunction $\Gamma^{\Phi}(C) \wedge \Phi$ is satisfiable.*

*Proof.* The proofs follow immediately from the associativity property of propositional logic. □

In principle, we may use an SMT solver to check the satisfiability of $\pi(l) \wedge \Phi$ for each leaf, in each tree. In practice, we reduce the dependence on SMT solvers and increase scalability by evaluating the robustness property during the encoding of the tree, where each internal node condition constraints a single feature $x_i$. For norm $p = \infty$, the leaf valuation $\pi(l)$ is satisfiable, if and only if for every node $n \in path(l)$ that refers to feature $x_i$, $|x_i - x_i'| \leq \epsilon$. For norm $p = 1$, a necessary condition for the satisfiability of $\pi(l)$, is that all features of $\vec{x}'$, $\sum_{i=1}^{m} d_i \leq \epsilon$, where:

$$d_i = \begin{cases} |x_i - x_i'|, & x_i \text{ appears in } path(l) \\ 0, & x_i \text{ does not appear in } path(l) \end{cases}$$

The pruning process removes paths where the given vector $\vec{x}'$ is "far" from the required thresholds by more than $\epsilon$, where the notion of distance is determined by the norm.

### 6.2 Parallelization

It is difficult to parallelize general SMT formulas efficiently. To increase scalability, we design our encoding in a manner that allows for parallel evaluation of gradient boosted classifiers. We do so by checking the robustness property separately for each class index. If all parallel evaluations are found robust, then the robustness property holds. Otherwise, there exists an assignment $\vec{x}$, and an index $q$, such that the robustness property does not hold, and $\vec{x}$ is a counter-example. The thread of class $q$ would discover this case and abort all other threads.

Formally, we do the following:

$$\begin{aligned}
&\forall \vec{x}' \quad : ||\vec{x} - \vec{x}'||_p < \epsilon \rightarrow \hat{C}(\vec{x}) = \hat{C}(\vec{x}') \\
\Leftrightarrow\ &\neg\exists \vec{x}' \quad : ||\vec{x} - \vec{x}'||_p < \epsilon \wedge \hat{C}(\vec{x}) \neq \hat{C}(\vec{x}') \\
\Leftrightarrow\ &\neg\exists \vec{x}' \quad : ||\vec{x} - \vec{x}'||_p < \epsilon \wedge \hat{C}(\vec{x}) \neq \arg\max_j \left(\hat{R}_j(\vec{x}')\right) \\
\Leftrightarrow\ &\neg\exists \vec{x}' \quad : ||\vec{x} - \vec{x}'||_p < \epsilon \wedge \exists q : \hat{R}_q(\vec{x}') > \hat{R}_{\hat{C}(\vec{x})}(\vec{x}') \\
\Leftrightarrow\ &\neg\exists \vec{x}', q : ||\vec{x} - \vec{x}'||_p < \epsilon \wedge \hat{R}_q(\vec{x}') > \hat{R}_{\hat{C}(\vec{x})}(\vec{x}')
\end{aligned}$$

Where the parameter $q$ is within $[1, c]$, and each thread verifies a different value of $q$. For example, if an input is classified as class $a$, we invoke $c - 1$ threads for classes $\{1, \ldots, c\} \setminus \{a\}$, where each thread tries to verify robustness with respect to a specific class.

## 7 Evaluation

We now introduce VERIGB (Verifier of Gradient Boosted models), which implements our approach in Python. VERIGB utilizes Z3 (De Moura and Bjørner 2008) as the underlying SMT solver. We used the `sklearn` (Buitinck et al. 2013) and `numpy` (Jones et al. 2001) packages to train models. We conducted the experiments on a VM with 36 cores, a CPU speed of 2.4 GHz, a total of 150 GB memory, and the Ubuntu 16.04 operating system. The VM is hosted by a designated server with two Intel Xeon E5-2680v2 processors (each processor is made of 28 cores at 2.4 Ghz), 260 GB memory, and Red Hat Enterprise Linux Server 7.3 operating system. For tractability, we capped the runtime of verifying the local robustness property by 10 minutes. We evaluated VERIGB using the following three datasets:

1. The *House Sales in King County (HSKC)* dataset containing $22K$ observations of houses sold in between May 2014 and May 2015 in King County, USA (housing 2018). Each observation has 19 house features, as well as the sale price.
2. The *Modified National Institute of Standards and Technology (MNIST)* dataset containing $70K$ images of handwritten digits (LeCun 1998). The images are of size 28 x 28 pixels, each with a grayscale value ranging from 0 to 255. The images are classified into 10 classes, one for each digit.
3. The *German Traffic Sign Recognition Benchmark (GTSRB)* dataset containing $50K$ colored images of traffic signs (Houben et al. 2013). The images are of size 32 x 32 pixels, each with three values (RGB) ranging from 0 to 255. The images are classified into 43 classes, one for each traffic sign.

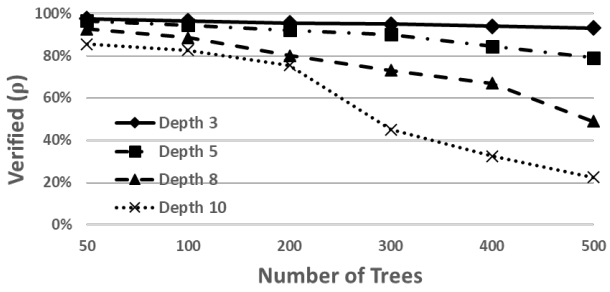


Figure 2: Universal robustness eveluation for $\epsilon = 160$ sq/ft, and $\delta = 100$K\$, and regressors with a similar score. Illustrating the attainable portion of robust observations $\rho$, varying the number of trees and the tree depth.

### 7.1 Regressor Evaluation

We start by demonstrating VERIGB's scalability to large gradient boosted regression models using the HSKC dataset. We trained regressors varying the learning rates in $\{0.1, 0.2, 0.3\}$, the number of trees between 50 and 500, and the tree depth in $\{3, 5, 8, 10\}$. All models have a similar score[2] that varies between 0.84 and 0.88. Then we randomly selected 200 observations and evaluated the $\rho$-universal $(\epsilon, \delta)$-robustness property with an $\epsilon$ value of 160 sq/ft, for the 6 numerical features that refer to square footage, and a $\delta$ value of 100K\$ in the price. Note that there were no timeouts (where it took the SMT solver more than 10 minutes to reach a decision) for models with less than 500 trees, and even with 500 trees we had only 16% timeouts.

Figure 2 illustrates the results for a learning rate of 0.1, while the results for other learning rates are similar. Notice that *(i)* robustness degrades as the number of trees increases. *(ii)* robustness seems to be negatively correlated with the tree depth. That is, a model trained with a tree depth of 3 is more robust than a depth of 5, which is more robust than 8 and 10.

### 7.2 Classifier Evaluation

Next, we demonstrate VERIGB's capability to verify the robustness of accurate classification models. We trained gradient boosted models for the MNIST and GTSRB datasets with a learning rate of 0.1. We varied the number of trees between 20 and 100, and the maximal tree depth between 3 and 20. The accuracy of said models varied between 87.9% and 97.3% for MNIST, and between 90% and 96.86% for GTSRB. We evaluated the $\rho$-universal $\epsilon$-robustness property with $\epsilon$ values of 1, 3, and 5. We randomly selected 20 images from each class in the training set (200 images for MNIST, and 860 images for GTSRB).

The illustration in Figure 1 is an artifact of this evaluation. Recall, that it shows two examples where the local adversar-

[2]The term score refers to the coefficient of determination $R^2$ of the prediction.

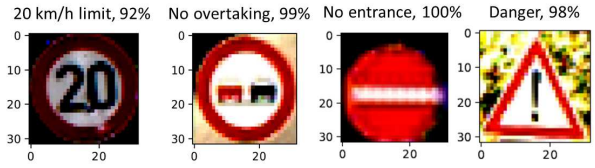


Figure 3: Examples of GTSRB images that satisfy the local adversarial robustness property for $\epsilon = 3$.

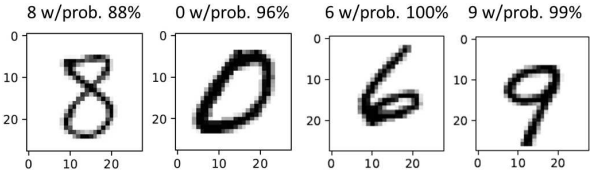


Figure 4: Examples of MNIST images that satisfy the local adversarial robustness property for $\epsilon = 3$.

ial robustness property does not hold for $\epsilon = 3$ for a model trained for the GTSRB dataset. In the first example, an "80" km/h speed limit sign is misclassified as a "30" km/h limit. In the second example, a "turn left" sign is misclassified as an "ahead only" sign. Alternatively, Figure 3 shows examples of signs that do satisfy the local adversarial robustness property for $\epsilon = 3$. That is, their classification would not change under any adversarial perturbation that changes each pixel's RGB values by at most 3.

Figure 4 shows examples of handwritten digits that satisfy the local adversarial robustness property for $\epsilon = 3$, for models trained for the MNIST dataset. Alternatively, Figure 5 shows two examples where the local adversarial robustness property does not hold. In the first example, an image of "1" is misclassified as "7". The second image is misclassified as "0" instead of "5" under very slight perturbation. These modifications are almost invisible to a human eye. Note that the model's confidence does not indicate robustness. E.g., in the first example the image has $95\%$ confidence to be classified as 1, while after applying the perturbation, it has $90\%$ confidence while being misclassified as 7.

**Scalability and limitations** Table 1 summarizes the results for selected models trained for the MNIST dataset. In the table, the abbreviations "T/O" and "C/E" stand for the portion of timeouts and counter-examples, respectively. Note that for a fixed tree depth, the portion of counter-examples found is negatively correlated with the model's accuracy. This is also true for a fixed number of trees. In this example, large models with 100 trees and high tree depth already exhibit a non-negligible portion of timeouts, indicating the limitations of VERIGB. Despite that fact, it successfully verifies highly accurate models for the MNIST dataset. We run similar experiments on models trained for the GTSRB dataset, with roughly similar results. Unlike MNIST, the portion of timeouts was only $1\%$, even for large models. As with MNIST, the portion of counter-examples varies

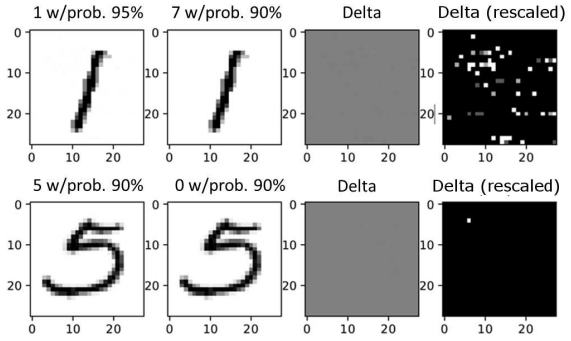


Figure 5: Examples of MNIST images that do not satisfy local adversarial robustness for $\epsilon = 3$. In the first row, an image of "1" is misclassified as "7". In the second row, an image of "5" is misclassified as "0". Observe in the third column (delta) that the applied changes are barely visible to the naked eye (delta of +/-3 in the range of 256 values per pixel per color). The fourth column highlights the modified pixels.

between $10\%$ and $22\%$. Finally, the ratio of robust images varies between $78\%$ and $88\%$.

**The effect of model structure on robustness** As a side-effect of this research, we noticed that certain configuration parameters tend to result in more robust models. Hereafter, we briefly discuss our observations. Table 2 summarizes selected results for models with a similar accuracy which is achieved by varying the number of trees, and the tree depth. As can be observed, models with smaller tree depth have a higher $\rho$ value. The results show that the tree depth has a potentially large impact on robustness. That is, increasing the tree depth leads to less robust results. Notice that tree depth similarly affects the robustness of regression models, as is clearly indicated in Figure 2.

It is interesting to mention, that tree depth also plays a role in the over-fitting problem of gradient boosted models. Models with large tree depth are more likely to suffer from over-fitting (Hastie, Tibshirani, and Friedman 2001). In our context, a small tree depth yields better robustness and is also easier to verify, making VERIGB attractive for practical use cases.

## 8 Related Work

Reliability and security are of increasing interest by the research community. Numerous works demonstrate the (lack of) security of popular machine learning models (Biggio, Fumera, and Roli 2014a; 2014b; Biggio et al. 2014). Others show methods to generate adversarial inputs to such models (Zhou et al. 2012). Thus, certifying that specific models are robust to adversarial inputs is an important research challenge. Indeed (Narodytska et al. 2018; Katz et al. 2017; Gehr et al. 2018), introduced methods for verifying robustness for various types of neural network models. The robustness of gradient boosted models is also of interest, but

| Depth | Trees | Accuracy | $\epsilon = 1$ | | | $\epsilon = 3$ | | | $\epsilon = 5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Verified ($\rho$) | T/O | C/E | Verified ($\rho$) | T/O | C/E | Verified ($\rho$) | T/O | C/E |
| 3 | 20 | 87.9 | 16.5% | 0% | 83.5% | 10% | 0% | 90% | 10% | 0% | 90% |
| 3 | 50 | 92.4 | 24% | 0% | 76% | 24% | 0% | 79% | 21% | 0% | 79% |
| 3 | 100 | 94.4 | 39.5% | 0.5% | 60% | 31.5% | 0.5% | 68% | 31.5% | 0.5% | 68% |
| 8 | 20 | 94.8 | 39.5% | 0% | 60.5% | 21% | 0% | 79% | 21% | 0% | 79% |
| 8 | 50 | 96.4 | 53.5% | 6% | 40.5% | 40% | 9.5% | 50.5% | 42.5% | 7% | 50.5% |
| 8 | 100 | 97 | 29% | 41.5% | 29.5% | 20% | 45% | 35% | 22% | 43.5% | 34.5% |
| 10 | 20 | 95.6 | 39.5% | 0% | 60.5% | 25% | 0% | 75% | 25% | 0% | 75% |
| 10 | 50 | 96.7 | 53% | 8.5% | 38.5% | 39.6% | 10.6% | 49.8% | 46% | 8.5% | 45.5% |
| 10 | 100 | 97.3 | 15% | 60% | 25% | 10.5% | 62.5% | 27% | 11.5% | 62.5% | 26% |

Table 1: MNIST dataset: Evaluating the attainable portion of robust observations $\rho$, for models with varying number of trees, tree depth, and $\epsilon$. The abbreviations "T/O" and "C/E" stand for the portion of timeouts and counter-examples, respectively.

| Depth | Trees | Accuracy | Verified ($\rho$) | T/O | C/E |
|---|---|---|---|---|---|
| 4 | 100 | 95.6 | 53% | 3% | 44% |
| 5 | 65 | 95.7 | 52% | 1% | 47% |
| 7 | 40 | 95.8 | 52% | 0.5% | 47.5% |
| 10 | 20 | 95.6 | 39.5% | 0% | 60.5% |
| 20 | 18 | 95.8 | 27.5% | 0.0% | 72.5% |

Table 2: MNIST dataset: Impact of boosted model's architecture on the attainable $\rho$ for the universal adversarial robustness property with $\epsilon = 1$.

existing works are focused on empirical evaluation (Leistner et al. 2009), or on training methods that increase robustness (Sun, Todorovic, and Li 2007), while our work is the first to certify gradient boosted models with formal and rigorous analysis.

Since our work is the first and only work that verifies gradient boosted model, we survey existing works that verify other machine learning models. In (Huang et al. 2017), the authors suggest an SMT based approach for verifying feed-forward multi-layer neural networks. They use a white box approach to analyze the neural network layer by layer and also apply a set of methods to discover adversarial inputs. Note that gradient boosted models are fundamentally different from neural networks and thus their method does not extend to such models. In (Katz et al. 2017), the authors describe a Simplex based verification technique, that is extended to handle the non-convex Rectified Linear Unit (ReLU) activation functions. Such activation is fundamental in modern neural networks and is not expressible with linear programming. The main disadvantage of that approach is its inability to scale up to large networks with thousands of ReLU nodes.

Alternatively, $AI^2$ (Gehr et al. 2018) uses "abstract transformers" to overcome the difficulty of formally describing non-linear activation functions. Safety properties such as robustness are then proved based on the abstract interpretation. The over-approximation that is inherent in the technique allows for scalable analysis. However, since they use abstractions, the counter-examples provided are not always real counter-examples, and thus a refinement process is required to end up with a concrete counter-example.

Finally, the authors of (Narodytska et al. 2018) adapt Boolean satisfiability to verify the robustness of Binarized Neural Networks (BNN). Specifically, they apply a counter-example-guided search procedure to check for robustness to adversarial perturbations. They verified BNN models for the MNIST dataset. In comparison, VERIGB verifies slightly more accurate gradient boosted models for the same dataset. Similarly, in (Ehlers 2017) the authors propose a method for verification of feed-forward neural networks. Their approach leverages piece-wise linear activation functions. The main idea is to use a linear approximation of the overall network behavior that can then be solved by SMT or ILP.

## 9 Conclusions and Future Work

Our work is the first to verify robustness to adversarial perturbations for gradient boosted models. Such models are among the most popular machine learning techniques in practice. Our work introduces a model verification tool called VERIGB that transforms the challenge of certifying gradient boosted regression and classification models into the task of checking the satisfiability of an SMT formula that describes the model and the required robustness property. This novel encoding is an important contribution of our work and includes formal correctness proofs as well as performance optimizations. Once we have such an (optimized) SMT formula, we check its satisfiability with a standard solver. The solver either proves the robustness property or provides a counter-example.

We extensively evaluated VERIGB, with 3 public datasets, and demonstrated its scalability to large and accurate models with hundreds of trees. Our evaluation shows that the classification's confidence does not provide a good indication of robustness. Further, it indicates that models with a small tree depth tend to be more robust even if the overall accuracy is similar. Such models are also known to suffer less from over-fitting. We believe that there may be an implicit correlation between robustness and good generalization, and leave further investigation to future work. Additionally, the counter-examples generated by VERIGB may be leveraged in the training phase of the gradient boosted models to optimize their robustness. However, we leave such usage for future work.

## References


Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press. 825–885.

Biggio, B.; Corona, I.; Nelson, B.; Rubinstein, B. I. P.; Maiorca, D.; Fumera, G.; Giacinto, G.; and Roli, F. 2014. *Security Evaluation of Support Vector Machines in Adversarial Environments*. Cham: Springer International Publishing. 105–153.

Biggio, B.; Fumera, G.; and Roli, F. 2014a. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering* 26(4):984–996.

Biggio, B.; Fumera, G.; and Roli, F. 2014b. Pattern recognition systems under attack: Design issues and research challenges. *IJPRAI* 28(7).

Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; Layton, R.; VanderPlas, J.; Joly, A.; Holt, B.; and Varoquaux, G. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122.

Caruana, R., and Niculescu-Mizil, A. 2006. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 161–168. New York, NY, USA: ACM.

Chapelle, O., and Chang, Y. 2010. Yahoo! learning to rank challenge overview. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14*, YLRC'10, 1–24. JMLR.org.

De Moura, L., and Bjørner, N. 2008. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, 337–340. Berlin, Heidelberg: Springer-Verlag.

Ehlers, R. 2017. Formal verification of piece-wise linear feed-forward neural networks. *CoRR* abs/1705.01320.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1):119 – 139.

Gehr, T.; Mirman, M.; Drachsler-Cohen, D.; Tsankov, P.; Chaudhuri, S.; and Vechev, M. T. 2018. Ai: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*.

Hastie, T.; Tibshirani, R.; and Friedman, J. 2001. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Houben, S.; Stallkamp, J.; Salmen, J.; Schlipsing, M.; and Igel, C. 2013. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

2018. House Sales in King County, USA. https://www.kaggle.com/harlfoxem/housesalesprediction/home.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017. Safety verification of deep neural networks. In Majumdar, R., and Kunčak, V., eds., *Computer Aided Verification*, 3–29. Cham: Springer International Publishing.

Jones, E.; Oliphant, T.; Peterson, P.; et al. 2001. SciPy: Open source scientific tools for Python.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In Majumdar, R., and Kuncak, V., eds., *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, 97–117. Springer.

LeCun, Y. 1998. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

Leistner, C.; Saffari, A.; Roth, P. M.; and Bischof, H. 2009. On robustness of on-line boosting - a competitive study. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 1362–1369.

Mason, L.; Baxter, J.; Bartlett, P.; and Frean, M. 1999. Boosting algorithms as gradient descent. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, 512–518. Cambridge, MA, USA: MIT Press.

Moosavi-Dezfooli, S.; Fawzi, A.; Fawzi, O.; and Frossard, P. 2017. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 86–94.

Narodytska, N.; Kasiviswanathan, S. P.; Ryzhyk, L.; Sagiv, M.; and Walsh, T. 2018. Verifying properties of binarized deep neural networks. In McIlraith, S. A., and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. AAAI Press.

Pulina, L., and Tacchella, A. 2010. An abstraction-refinement approach to verification of artificial neural networks. In *Proceedings of the 22Nd International Conference on Computer Aided Verification*, CAV'10, 243–257. Berlin, Heidelberg: Springer-Verlag.

Pulina, L., and Tacchella, A. 2012. Challenging smt solvers to verify neural networks. *AI Communications* 25(2):117–135.

Sun, Y.; Todorovic, S.; and Li, J. 2007. Increasing the robustness of boosting algorithms within the linear-programming framework. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology* 48(1):5–20.

Viola, P., and Jones, M. 2001. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, I–511–I–518 vol.1.

Yang, B.; Yan, J.; Lei, Z.; and Li, S. Z. 2015. Convolutional channel features for pedestrian, face and edge detection. *CoRR* abs/1504.07339.

Zhang, Y., and Haghani, A. 2015. A gradient boosting method to improve travel time prediction. In *Transportation Research Part C Emerging Technologies*, volume 58.

Zhou, Y.; Kantarcioglu, M.; Thuraisingham, B.; and Xi, B. 2012. Adversarial support vector machine learning. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, 1059–1067. New York, NY, USA: ACM.