Hardware Trojan Detection using Shapley Ensemble Boosting

Zhixin Pan and Prabhat Mishra
University of Florida, Gainesville, Florida, USA

Abstract

Due to globalized semiconductor supply chain, there is an increasing risk of exposing system-on-chip designs to hardware Trojans (HT). While there are promising machine Learning based HT detection techniques, they have three major limitations: ad-hoc feature selection, lack of explainability, and vulnerability towards adversarial attacks. In this paper, we propose a novel HT detection approach using an effective combination of Shapley value analysis and boosting framework. Specifically, this paper makes two important contributions. We use Shapley value (SHAP) to analyze the importance ranking of input features. It not only provides explainable interpretation for HT detection, but also serves as a guideline for feature selection. We utilize boosting (ensemble learning) to generate a sequence of lightweight models that significantly reduces the training time while provides robustness against adversarial attacks. Experimental results demonstrate that our approach can drastically improve both detection accuracy (up to 24.6%) and time efficiency (up to 5.1x) compared to state-of-the-art HT detection techniques.

ACM Reference Format:
Zhixin Pan and Prabhat Mishra. 2023. Hardware Trojan Detection using Shapley Ensemble Boosting. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3566097.3567920

1 Introduction

A vast majority of semiconductor companies rely on global supply chain to reduce design cost and meet time-to-market deadlines. The benefit of globalization comes with the cost of security concerns. A typical automotive System-on-Chip (SoC) consists of multiple Intellectual Property (IP) cores, some of these cores may come from potentially untrusted third-party suppliers. An attacker may be able to introduce malicious implants, popularly known as Hardware Trojans (HT). HT is a malicious modification of the target integrated circuit (IC) with two critical parts, trigger and payload. The trigger are typically created using a combination of rare events (such as rare signals or rare transitions) to stay hidden during normal execution. The payload represents the malicious impact on the target design, commonly resulting in information leakage or erroneous execution. When the trigger is activated, the payload enables the malicious activity. For example in Figure 1, when the output of the trigger logic (grey AND gate) is true, the output of the payload (grey XOR gate) will invert the expected output. It is vital to detect HTs to enable trustworthy computing using modern SoCs.

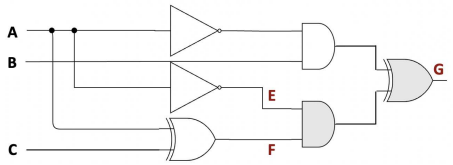

Figure 1: *An example hardware Trojan constructed by a trigger logic (gray AND gate). Once the trigger condition is satisfied, the payload (gray XOR gate) will invert the expected output.*

There are many promising approaches for machine learning (ML) based HT detection [3, 10, 14]. However, they have three inherent limitations. First, they provide only detection results without interpreting them in a human understandable way. Next,they focus on extracting 'features' from given dataset, but the feature selection relies on expert knowledge without any established guidelines. Finally, existing efforts focus on generating complicated models to improve the detection accuracy, which may introduce unacceptable training cost for parameter tuning.

In this paper, we propose an efficient HT detection approach based on *Shapley Ensemble Boosting* (SEB) which addresses the above challenges. It efficiently combines the Shapley analysis (SHAP) and boosting to enable an explainable, fast and robust detection model. Specifically, this paper makes the following major contributions:

- SEB provides an explainable ML framework. It exploits SHAP to generate the spectrum of impact of each feature towards the model output. It explains the decision process and serves as a guideline for feature selection.
- SEB a provides a fast and robust boosting framework. It explores ensemble model by combining a sequence of lightweight models to generate a powerful classifier.
- Extensive evaluation shows significant improvement in both detection accuracy (up to 24.6%) and time efficiency (up to 5.1x) compared to state-of-the-art approaches.

The paper is organized as follows. Section 2 surveys related efforts. Section 3 describes our Trojan detection framework. Section 4 presents experimental results. Finally, Section 5 concludes the paper.

This work was partially supported by the NSF grant CCF-1908131.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567920

2 Background and Related Work

We first survey related efforts on hardware Trojan detection. Next, we provide background on boosting and Shapley values.

2.1 Related Work for HT Detection

There are many promising efforts for hardware Trojan detection that can be broadly classified into three categories: side-channel analysis, simulation-based validation, and ML-based HT detection. Side-channel analysis focuses on the difference in side-channel signatures (such as power, path delay, etc.) between the expected (golden specification) and actual (Trojan-inserted implementation) values [4, 6, 9]. A major drawback in side-channel analysis is that it is difficult to detect the negligible side-channel difference caused by a tiny Trojan since it can easily hide in process variation and environmental noise. Simulation-based HT detection is not affected by any process/noise variations since it relies on test patterns to activate HTs. MERO [2] proposed a statistical test generation scheme to activate the rare trigger. COTD [13] proposed a static HT detection approach based on testability analysis. TARMAC [7] utilizes maximal clique sampling to generate efficient test patterns. it is a fundamental challenge to activate an extremely rare trigger without trying all possible (exponential) input sequences.

ML-based HT detection does not have any of the above disadvantages. It focuses on extracting 'features' from a large amount of historical data to train models to perform experience-based predictions. Hasegawa et al. [3] proposed a static HT detection technique using random forest. A similar neural network based approach has been explored by [14]. Recently, Pan et al. [10] use reinforcement learning for improved feature selection to reduce the false positive rate. There are several major drawbacks of existing ML-based HT detection methods. First, There is no clear guideline for feature selection, existing feature selection work for HT detection are heavily determined by human expert knowledge. Next, due to the black-box nature of most ML models, existing approaches lack the transparency, and are unable to interpret their predictions in a meaningful way. Finally, ML algorithms themselves are vulnerable towards adversarial attacks, making their usage in security-focused domain unreliable. In this paper, we address these challenges by combining Shapley value analysis with boosting framework as described in Section 3.

2.2 Ensemble Boosting

Boosting is a machine learning technique to improve the accuracy of predictive models. It creates stronger models by combining multiple weaker models, such as decision trees. The individual models are trained sequentially, with each model compensating for the errors of the previous model. The final prediction is made by combining the predictions of all the individual models. Boosting can be used for regression and classification tasks and is a powerful tool for dealing with complicated tasks. It is also relatively resistant towards overfitting problem, and achieves high levels of accuracy without sacrificing generalization. This can also lead to fast prediction since multiple models can work in parallel at run-time. Figure 2 shows an overview of a boosting framework.

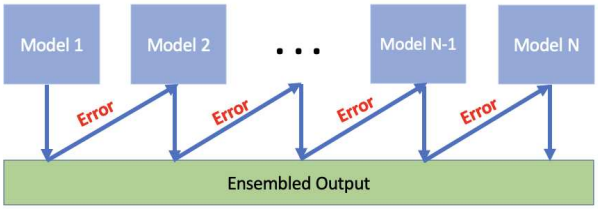

Figure 2: *The ensemble consist of a set of weak classifiers. Subsequent models focus on fixing the weakness of previous models. The final decision is based on the overall voting result.*

2.3 Shapley Values

ML has shown its potential in security domain tasks. However, due to their black-box nature, no further information aside from detection result can be provided by the ML models. What's worse, security practitioners gain no clue for incorrect predictions. This lack of transparency make people hesitate to widely adopt them in safety-critical domains. In our work, we address this challenge by utilizing explainable ML. Specifically, explanation techniques aim to illustrate what is the major reason for model transferring certain input into its prediction. This often involves identifying a set of important features that make key contributions to the forward pass of model. In our proposed method, we utilize Shapley value analysis to provide the contribution measurement.

The concept of Shapley values (SHAP) is borrowed from the cooperative game theory [11]. It is used to fairly attribute a player's contribution to the end result of a game. SHAP capture the marginal contribution of each player to the final result. Formally, we can calculate the marginal contribution of the i-th player in the game by:

$$\phi_i = \sum_{S \subseteq N/\{i\}} \frac{|S|!(M-|S|-1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)] \quad (1)$$

where the total number of players is $|M|$. S represents any subset of players that does not include the i-th player, and $f_x(\cdot)$ represents the function to give the game result for the subset S. Intuitively, SHAP is a weighted average payoff gain that player i provides if added into every possible coalitions without i. We will show how Shapley value analysis is applied in the task of HT detection in Section 3.3.

3 Shapley Ensemble Boosting for Hardware Trojan Detection

Our proposed approach enables a synergistic integration of Shapley value analysis (SHAP) and boosting for efficient HT detection. Figure 3 shows an overview of our proposed method that consists of five major tasks. The first task performs *Data Sampling* from given sources. We randomly sample benchmarks from the entire pool and extract a subset of features from them. The second task performs *Model Training* that

trains a lightweight classifier based on sampled data. The trained classifier is immediately tested to record correct/incorrect predictions. The third task performs *Shapley Analysis* to figure out the importance ranking of each feature. The fourth task performs *Weight Adjustment* to adjust the weights for each benchmark as well as each feature in the list. These four tasks repeat for sufficient number of iterations until reaching convergence. The final task uses the well-trained framework to perform an *Ensemble Prediction*. For any given testing benchmark, all the classifiers generated during this process will produce their own prediction, and the overall voting determines if the benchmark has any hardware Trojan. The remainder of this section describes these tasks in detail.

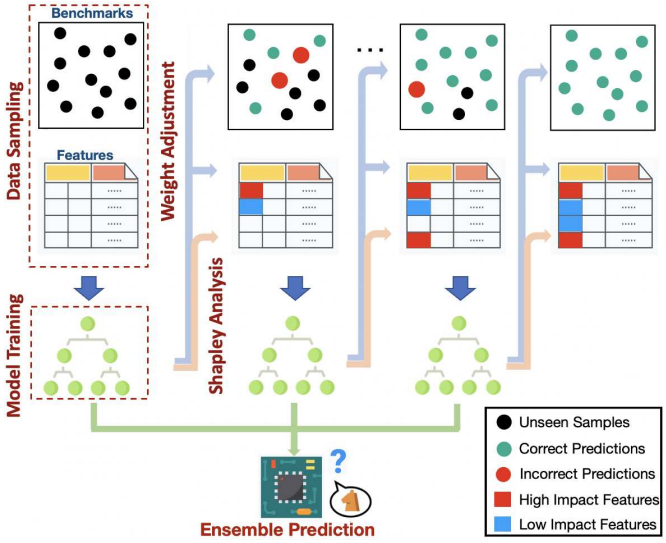

Figure 3: *An overview of our proposed Shapley ensemble boosting framework for hardware Trojan detection.*

3.1 Data Sampling

The key idea of boosting framework is to train a sequence of lightweight classifiers and adopt the aggregation result as the output. To improve the efficiency, training cost for each individual classifier should be restricted. To address this, we perform data sampling before model training at each iteration. The sampling considers two aspects: benchmarks and features. At each iteration, only a subset of benchmarks and features are fed into the model. To enable a comprehensive evaluation, we collect data from benchmarks from both Trust-Hub [12] and ISCAS-89 [1]. As for features, we adopt the idea from [3], where the authors proposed 51 important features for HT detection, including but not limited to the number of logic-gate *fan_ins*, *flip-flops*, *multiplexers* as well as *loops* in netlists. These features are intuitively related with malicious implants. For example, in case of combinational circuit triggers, the number of *fan_ins* tends to become large for extremely rare triggers. We have also included the total number of nets and cells into consideration since they provide the overall statistics. All the above features are based on static analysis. To better inspect the property of benchmarks, we also simulate these benchmarks with test vectors from [9] to get their dynamic power changes as well as total number of rare switches. The complete list of all 55 candidate features are shown in Table 1. Initially, every benchmark has equal chance to be sampled. Similarly, every feature has equal probability of getting sampled initially.

Table 1: *The list of all candidate features* ($1 \leq x \leq 5$)

Features	Description
Nets	Total # of nets
Cells	Total # of cells
fan_in_x	# fanins up to x-level away from the PI/PO.
in_FF_x	# flip-flops up to x-level away from the PI.
out_FF_x	# flip-flops up to x-level away from the PO.
in_MUX_x	# MUXs up to x-level away from the PI.
out_MUX_x	# MUXs up to x-level away from the PO.
in_loop_x	# up to x-level loops.
out_loop_x	# up to x-level loops.
in_const_x	# constants up to x-level away from the PI.
out_const_x	# constants up to x-level away from the PO.
in_pin	The level to the PI from the nearest net.
out_pout	The Level to the PO from the nearest net.
{ in, out }_FF	minimum level to any flip-flop from the PI/PO.
{ in, out }_MUX	minimum level to any MUX from the PI/PO.
Rare Switches	# of total rare switches during simulation.
Dynamic Power	dynamic power change during simulation (mW).

3.2 Model Training

Once the subset of sampled data is obtained, the model training process starts. In our framework, each lightweight classifier is chosen to be a *decision tree* (DT). DT is a supervised learning approach based on a tree structure. It predicts the class of the target by traversing from the root to the leaf of the tree. We compare the values of the node attributes with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the children nodes until the leaf node is reached. We follow the traditional steps to train DTs, and specifically, we utilize CART [5] method to generate the trees. To enable fast training speed, the maximum depth of each DT is kept less than 6. We also utilize both L1 (Lasso Regression) and L2 (Ridge Regression) regularization to prevent the model from overfitting. Once the training finishes, the trained DT is used to detect hardware Trojans, and we record the correct/incorrect prediction for all the testing samples. These records are further analyzed in the next step.

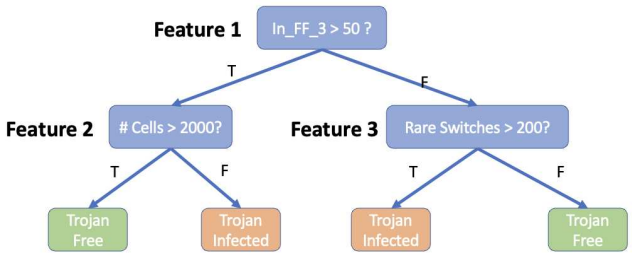

Figure 4: *Example decision tree that classifies using 3 features*

3.3 Shapley Analysis

To apply SHAP in ML tasks, we can assume features as the 'players' in a cooperative game. SHAP is a local feature attribution technique that explains every prediction from the model as a summation of each individual feature contributions. Assume a decision tree is built with 3 different features for HT detection as shown in Figure 4. To compute SHAP values, we start with a null model without any independent features. Next, we compute the payoff gain as each feature is added to this model in a sequence. Finally, we compute average over all possible sequences. Since we have 3 independent variables here, we have to consider 3!=6 sequences. Specifically, the computation process for the SHAP value of the first feature is presented in Table 2.

Table 2: *Marginal contributions of the first feature for the model.*

Sequences	Marginal Contributions
1,2,3	$\mathcal{L}(\{1\}) - \mathcal{L}(\emptyset)$
1,3,2	$\mathcal{L}(\{1\}) - \mathcal{L}(\emptyset)$
2,1,3	$\mathcal{L}(\{1,2\}) - \mathcal{L}(\{2\})$
2,3,1	$\mathcal{L}(\{1,2,3\}) - \mathcal{L}(\{2,3\})$
3,1,2	$\mathcal{L}(\{1,3\}) - \mathcal{L}(\{3\})$
3,2,1	$\mathcal{L}(\{1,2,3\}) - \mathcal{L}(\{3,2\})$

Here, $\mathcal{L}$ is the loss function. The loss function serves as the 'score' function to indicate how much payoff currently we have by applying existing features. For example, in the first row, the sequence is $1, 2, 3$, meaning we sequentially add the first, second, and the third features into consideration for classification. $\emptyset$ stands for the model without considering any features, which in our case is a random guess classifier, and $\mathcal{L}(\emptyset)$ is the corresponding loss. Then by adding the first feature into the scenario, we use $\{1\}$ to represent the dummy model that only uses this feature to perform prediction. We again compute the loss $\mathcal{L}(\{1\})$. $\mathcal{L}(\{1\}) - \mathcal{L}(\emptyset)$ is the marginal contribution of the first feature for this specific sequence. We obtain the SHAPs for the first feature by computing the marginal contributions of all 6 sequences and taking the average. Similar computations happen for the other features. The SHAP values are crucial indicator of their impact towards model decisions, as explored in the next section.

3.4 Weight Adjustment

The weight adjustment step aims at tuning the probabilities of selecting benchmarks and features in the next iteration of training. Intuitively, while adjusting weights, we need to follow three guidelines. First, for incorrectly predicted benchmarks, their weights should be increased. Next, for high-impact features from incorrectly predicted benchmarks, their weights should be decreased. Finally, for high-impact features from correctly predicted benchmarks, their weights should be increased. In our framework, we always normalize the weight values at the start of each iteration, therefore, there is no need to decrease the weights for correctly classified benchmarks.

Formally, we denote $X_i \in \mathcal{D}$ as the i-th benchmark from the entire dataset $\mathcal{D}$. X_i's corresponding label is denoted as y_i, where $y_i = 0$ represents for a Trojan-free benchmark and $y_i = 1$ for a Trojan-infected one. The ML model's prediction probability for this benchmark is denoted as $\hat{y}_i$. The weight of X_i is recorded as w_i^b. As for features, we denote $F_j \in \mathcal{F}$ as the j-th feature from the candidate feature list (Table 1). The weight of F_j is represented by w_j^f. Especially, we use F_{ij} to represent the value of F_j for the specific instance X_i. The remainder of this section briefly describe the weight adjustment procedure for benchmarks as well as features.

Weight Adjustment for Benchmarks: The goal of adjusting benchmark weights is to maximize the chance of sampling these hard-to-fit samples in the next iteration. To accelerate the convergence speed, benchmarks with larger prediction error should take higher priority. Therefore for misclassfied benchmark X_i, we should have

$$\Delta w_i^b \propto \mathcal{L}(y_i, \hat{y}_i)$$

where $\mathcal{L}$ is the loss function measuring the extent of prediction error (mentioned in Section 3.3). In binary classification task, $\mathcal{L}$ is selected as the cross-entropy,

$$\mathcal{L}(y_i, \hat{y}_i) \triangleq (y_i \, log \, \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i))$$

which leads to

$$\Delta w_i^b \propto (y_i \, log \, \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i))$$
$$\Rightarrow w^b_{i_{(t+1)}} = w^b_{i_{(t)}} + \alpha(y_i \, log \, \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i))$$

where t represents for the t-th iteration, α is a hyperparameter which affects the learning rate. In our framework, we artificially limit the step size to be small, so that the parameters are more likely to converge to the optimal values, thereby providing better overall performance for the model.

Weight Adjustment for Features: The adjustment for feature weights are similar to benchmark weights but differs in three aspects. First, the extent of adjustment is measured by the contribution of the feature towards the model output. The more contribution it possess, larger the extent of adjustment. Next, the correctness should be taken into consideration. As outlined earlier, in case of high-impact features from incorrectly predicted benchmarks, their weights should be decreased, since it is a dominating factor contributing to an incorrect prediction. Finally, for each feature, their contribution varies from benchmark to benchmark, therefore, the computation should be an overall summation among all tested samples. We measure the contribution by SHAP. We use $SHAP(F_{ij})$ to denote the SHAP value of F_{ij}. Then

$$\Delta w_j^f \propto \sum (\pm w_i^b \, \mathcal{L}(y_i, \hat{y}_i) \, SHAP(F_{ij}))$$

where $\pm$ is positive for correct prediction and vice versa. Notice Δw_i^f is a weighted summation, since we should also take the benchmarks' weights into consideration. Large benchmark weights indicate their necessity. Their corresponding features are emphasized than the others. Therefore, we have

$$w^f_{j_{(t+1)}} = w^f_{j_{(t)}} + \beta \sum (\pm w^b_{i_{(t+1)}} \, \mathcal{L}(y_i, \hat{y}_i) \, SHAP(F_{ij}))$$

where t represents the t-th iteration, and β is another hyper-parameter. We follow the same criteria for tuning β as that for α. The above two procedures are used to adjust and normalize the weight values for next round of training iteration. The model training and weight adjustment steps continue until reaching convergence.

Algorithm 1: Detect with Shapley Ensemble Boosting

Input : Benchmark Dataset($\mathcal{D}$), Feature Set ($\mathcal{F}$), Instance (s), learning rate (α, β), epochs (k)
Output : Ensemble Model $\mathcal{T}$, Prediction res

1 Initialize:
2 $\mathcal{T} = \emptyset$, N = sizeof ($\mathcal{D}$), M = sizeof ($\mathcal{F}$), $t = 0$
3 For each $X_i \in \mathcal{D}$, $w^b_{i_{(t)}} = \frac{1}{N}$
4 For each $F_i \in \mathcal{F}$, $w^f_{i_{(t)}} = \frac{1}{M}$
5 **repeat**
6 Random Sample $\mathcal{D}' \subset \mathcal{D}, \mathcal{F} \subset \mathcal{F}'$ ▷ Data Sampling
7 $T_{(t)}$ = CART($\mathcal{D}', \mathcal{F}'$) ▷ Model Training
8 **for** *each* $X_i \in \mathcal{D}'$ **do**
9 $\hat{y}_i = T(X_i)$
10 $\mathcal{L}(y_i, \hat{y}_i) = (y_i \, log \, \hat{y}_i + (1 - y_i) log(1 - \hat{y}_i))$
11 $w^b_{i_{(t+1)}} = w^b_{i_{(t)}} + \alpha \mathcal{L}(y_i, \hat{y}_i)$ ▷ Weight Adjustment
12 **for** *each* $F_j \in \mathcal{F}'$ **do**
13 Compute Shapley Values $SHAP(F_{ij})$
14 $w^f_{j_{(t+1)}} = w^f_{j_{(t)}} + \beta \sum (\pm w^b_{i_{(t+1)}} \mathcal{L}(y_i, \hat{y}_i) SHAP(F_{ij}))$
15 $\mathcal{T} = \mathcal{T} \cup \{T_{(t)}\}$
16 $t = t + 1$
17 **until** $t \geq k$ *or reaching convergence*;
18 $res = \mathcal{T}(s)$ ▷ Ensemble Prediction
19 Return $\mathcal{T}, res$

3.5 Ensemble Prediction

The overall ensemble prediction is the voting result of all predictions from each tree. There is no need to assign weights to each tree. Intuitively, if every tree has same weight, for benchmarks misclassified by the first tree, even if it is correctly classified by the second tree, then the overall voting result is still fifty-fifty. In fact, by increasing the weights of misclassified sample and significantly decreasing the most influential feature causing misclassification, chances for subsequent models to vote for incorrect prediction is extremely low. If we decrease the weights for the first several trees that make mistakes, then the correctly predicted benchmarks are also affected. These benchmarks are hardly selected by subsequent models, while the weights of models voting for their truth label are diminished. Algorithm 1 summarizes the above discussion to highlight the major steps in our HT detection framework. Specifically, Line 6 denotes *Data Sampling* (Section 3.1). Line 7 covers *Model Training* (Section 3.2). Line 8-14 performs *Shapley Analysis* (Section 3.3) and *Weight Adjustment* (Section 3.4) for each model. Finally, Line 18 performs *Ensemble Prediction* (Section 3.5) to detect if there is any hardware Trojan in the test benchmark.

4 Experiments

In this section, we evaluate the effectiveness of our hardware Trojan detection framework. First, we describe the experimental setup. Next, we compare with state-of-the-art approaches.

4.1 Experimental Setup

To enable fair comparison with existing approaches, we use the same benchmarks as [2, 3, 10] from Trust-Hub [12] and ISCAS-89 [1]. Randomly sampled benchmarks are injected with 1000 HTs. For each benchmark, we record static features including total number of nets and cells, netlist features introduced in [3], along with simulation-based features (rare switches, dynamic power change). The list of all 55 candidate features are shown in Table 1. For counting the number of rare switches, we preserve the same parameter configuration applied in those papers, where rareness threshold is set to 0.1.

The code for benchmark parsing and identification of rare nodes is written in C++. The machine learning model was conducted trained/tested on a host machine with Intel i7 3.70GHz CPU, 32 GB RAM and RTX 2080 256-bit GPU. We choose Python (3.6.7) code using scikit-learn (1.1.1) with cudatoolkit (10.0) to implement the GPU acceleration framework. The maximum number of epochs is set to 1000 during the training phase. We compare in terms of detection accuracy as well as time efficiency between the following methods:

- **RFC:** State-of-the-art statistical Trojan detection at gate-level using Random Forest (RF). [3].
- **CNN:** State-of-the-art Trojan detection using Convolution Neural Network (CNN) [14].
- **TGRL:** State-of-the-art test generation method for Trojan detection using reinforcement learning [10].
- **SEB:** Our proposed Shapley ensemble boosting framework for hardware Trojan detection.

In this paper, we denote a "Trojan-infected" case as *positive*, and successfully detecting a Trojan-infected benchmark is recorded as "True-Positive", and vice versa.

4.2 HT Detection Performance

We consider the following four metrics, where tp, tn, fp and fn are the number of true positive, true negative, false positive and false negative, respectively. Recall is a measure of a classifier's exactness, while precision is a measure of a classifiers completeness, and F1 score is the harmonic mean of recall and precision.

- **Accuracy:** $\frac{tp+tn}{tp+tn+fp+fn}$
- **Recall:** $\frac{tp}{tp+fn}$
- **Precision:** $\frac{tp}{tp+fp}$
- **F1 Score:** $\frac{tp}{tp+\frac{1}{2}(fp+fn)}$

Table 3: *Comparison of HT detection performance using accuracy (Acc), recall (Rec), precision (Pre) and F1 score (F1).*

	RFC [3]				CNN [14]				TGRL [10]				SEB (Proposed Approach)				
Bench	**Acc**	**Rec**	**Pre**	**F1**	**Acc**	**Rec**	**Pre**	**F1**	**Acc**	**Rec**	**Prec**	**F1**	**Acc**	**Rec**	**Pre**	**F1**	**impr/TGRL**
c2670	83.1%	0.87	0.89	0.88	90.7%	0.90	0.90	0.90	96.2%	0.97	0.94	0.96	100.0%	1.0	1.0	1.0	3.8%
c5315	75.4%	0.78	0.83	0.81	87.6%	0.85	0.88	0.86	91.4%	0.92	0.91	0.92	100.0%	1.0	1.0	1.0	8.6%
c6288	64.5%	0.68	0.63	0.65	80.5%	0.85	0.79	0.85	88.8%	0.89	0.85	0.87	99.8%	0.99	0.99	0.99	11.0%
c7552	77.2%	0.74	0.79	0.76	84.9%	0.81	0.86	0.83	91.2%	0.89	0.91	0.90	100.0%	1.0	1.0	1.0	8.8%
s13207	78.5%	0.77	0.79	0.78	90.4%	0.91	0.92	0.92	95.6%	0.94	0.95	0.95	100.0%	1.0	1.0	1.0	4.4%
s15850	68.8%	0.65	0.73	0.68	83.0%	0.75	0.86	0.80	92.7%	0.93	0.95	0.94	99.8%	0.99	0.99	0.99	7.1%
s35932	73.1%	0.78	0.53	0.63	75.5%	0.72	0.76	0.74	83.6%	0.88	0.81	0.84	99.9%	0.97	0.99	0.98	16.3%
AES-T100	85.9%	0.93	0.79	0.85	89.2%	0.84	0.86	0.85	96.9%	0.97	0.97	0.97	100.0%	1.0	1.0	1.0	3.1%
AES-T200	79.3%	0.88	0.73	0.79	90.2%	0.85	0.92	0.88	95.8%	0.98	0.91	0.94	99.9%	1.0	1.0	1.0	4.1%
AES-T1000	67.2%	0.84	0.63	0.72	80.5%	0.72	0.76	0.74	90.1%	0.95	0.95	0.95	99.9%	1.0	1.0	1.0	9.8%
Average	**75.3 %**	**0.79**	**0.73**	**0.76**	**85.3%**	**0.82**	**0.85**	**0.83**	**92.2%**	**0.93**	**0.91**	**0.92**	**99.9%**	**0.99**	**1.0**	**1.0**	**6.1**

Table 3 compares the performance of our approach (SEB) with the existing methods. We present the HT detection performance of all four methods on various benchmarks using accuracy (Acc), recall (Rec), precision (Pre), and F-1 score (F1). Each row stands for one specific benchmark. The average values of evaluation are also plotted in Figure 5. The RFC model achieves 75% accuracy, and CNN model achieves 85%. Their performances falls behind TGRL and our proposed method. This observation is supported by the intrinsic of models applied in these methods. RFC utilizes random forest (RF) which consists of a forest of decision trees (DTs) to make aggregation decisions. It is similar to our boosting framework but RFC does not perform any weight adjustment for any benchmarks or features. As a result, RFC has the lowest value of recall among all methods. Intuitively, a low *recall* score inflicts high proportion of false negatives. While in our proposed method, after each training iteration, each misclassified sample is marked with high sampling weights and becomes a major focus for subsequent decision trees. As for CNN, it is a well-known ML model for processing *computer vision* tasks. They are good at extracting linear shift invariants from images, but not specifically designed for HT detection. Our proposed method achieves 99.9% accuracy with 1.0 F1 score, which is up to 24.6% improvement over existing efforts.

There are several reasons for our proposed method's superior performance over state-of-the-art methods. The sophisticated feature selection strategy guided by the SHAP analysis helps to identify the most influential features in early stages, which helps to avoid the disturbance caused by redundant features. SEB is an ensemble model which make predictions based on voting of all sub-models. This strategy significantly reduces the bias. Moreover, the sequential training strategy allows each sub-model concentrating on the previously incorrect classified observations. The errors are gradually reduced through iterations.

4.3 Explainability Analysis

In this section, we demonstrate the transparency and explainability for the proposed HT detection approach by SHAP. In ML, the task of classification commonly boils down to compute a separator and check which side the sample falls in.

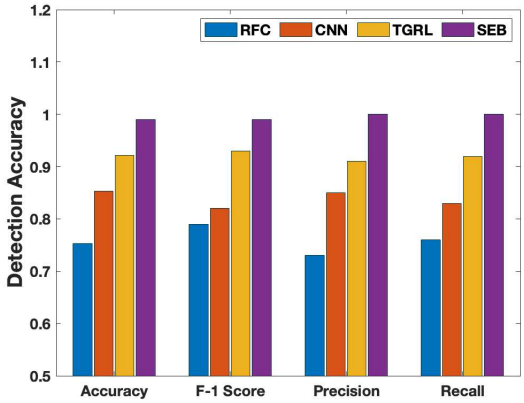

Figure 5: *Comparison of average HT detection performance by various methods using accuracy, precision, recall, and F-1 score.*

Since HT detection is a binary classification, the task is reduced to computing a threshold value and compare with the model output. It is classified as negative (Trojan-Free) if it lies in the left (smaller), positive (Trojan-infected) otherwise.

Figure 6 shows the waterfall plot of SHAP values from a pair of samples, (a) a true negative sample and (b) a true positive one. In our case, the threshold value is 2.192. The waterfall plot clearly demonstrate the contribution of each feature and how they affect the decision. The plus or minus sign illustrate whether the specific feature is supporting the sample to be positive (red bars), or voting for the negative (blue bars). The SHAP values along with each bar show their exact impact, and the summation of all SHAP values is compared with the threshold to give the final decision. As we can see from the figure, in both (a) and (b), rare switches and dynamic power change are among the most important features. This is reasonable since HTs are more likely to be designed with rare signals as triggers, and the change of dynamic power is also an important indicator of Trojan injection. Notice there are 55 candidate features as we mentioned in Table 1, but only the top 8 features are shown in our plot since the rest of them barely provides any contribution (< 0.01). In fact, after the first several iterations of training, these redundant features' weights are significantly reduced, and therefore are less likely to be selected by subsequent models. This strategy drastically reduced the training time, as we will show in Section 4.4.

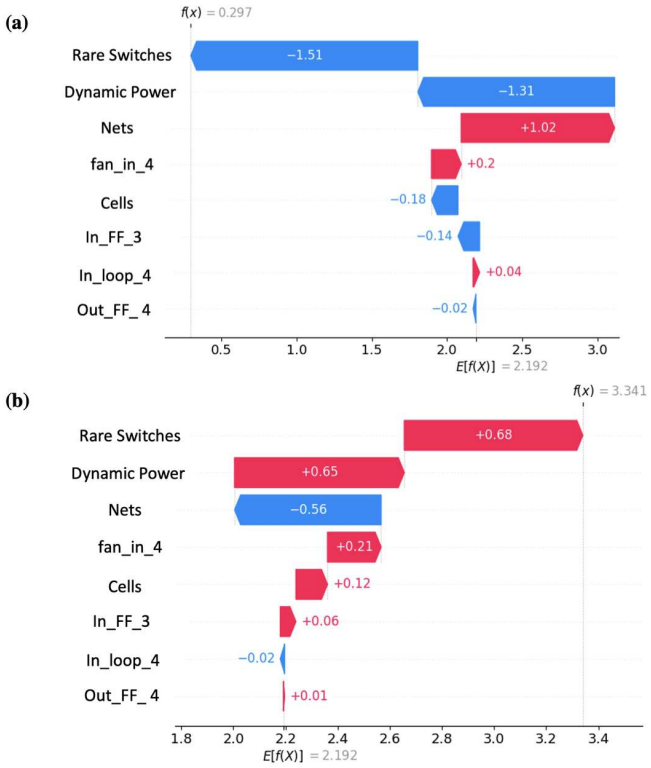

Figure 6: *The SHAP values for two example cases. (a) A Trojan-free benchmark, and (b) A Trojan-infected benchmark. The SHAP values clearly illustrate the major features.*

Waterfall plots only show insights for individual samples. To provide a better visual illustration, we generate the decision plot in Figure 7 for 10 random samples (5 positives and 5 negatives) in the dataset. In this figures, each polyline represents one single sample. The x-axis represents the model's output value, and the plot is centered on the x-axis at the threshold value. The y-axis lists the model's features ordered by descending importance. For each line, a traverse from the bottom to the top clearly displays how each features are contributing to the decision. Starting at the bottom, every sample converges at the threshold value. When moving bottom-up, SHAP values for each feature are computed and added to the base value. When reaching at the top of the plot, each line strikes the x-axis at its corresponding model output, and this value determines the prediction result. Clearly, the more features we take into consideration, the more two clusters of lines diverge from each other. Finally, five lines hit on the left side (negatives) of the central gray line, and the other five on the right (positives). Notice in this plot, we show the impacts of the 12 most influential features, but in fact the bottom 4 features barely make any contribution. For every polyline, after traversing the bottom 4 contributions, the values are still only slightly deviated from the threshold. It is not until we take the top 8 features into consideration, the lines in two classes start to diverge. This observation matches our analysis from previous plots.

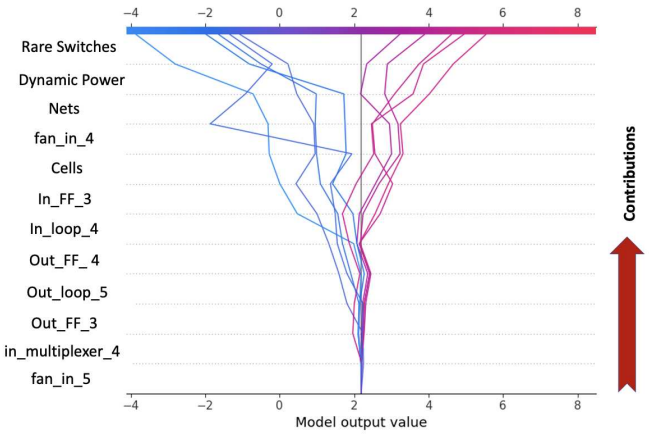

Figure 7: *The decision plot for 10 random samples.*

SHAP clearly illustrate the decision process of the proposed framework for HT detection, and as we observed from Figure 6 and Figure 7, the total number of rare switches are considered as the most influential features for HT detection. To ensure the credibility of our model, we need to guarantee the validity of this feature. The scatter plot in Figure 8 shows the distribution of the rare switches in the entire dataset and its dependence with respect to model scale (number of cells). In this plot, each dot is a single benchmark. The x-axis represents the number of rare switches, and the y-axis is the corresponding Shapley value. When the value of rare switches are very low (< 60), they barely make contribution to the model (low Shapley values), since in this case they do not provide any meaningful information. However, when sufficient number of rare switches are recorded, larger the value, more contributions are observed to support model's prediction.

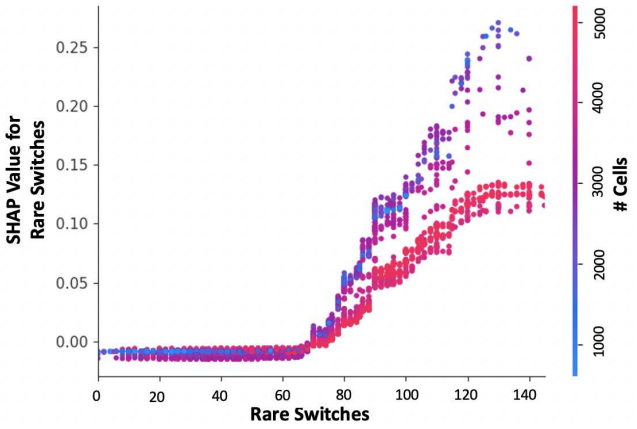

Figure 8: *The dependency plot.*

4.4 Efficiency Analysis

Table 4 compares the time efficiency for all four methods. The first row lists the name of methods, while the next two rows provide the average training time and testing time respectively. In the last three columns, we show the time improvement provided by our approach compared to the others.

Table 4: *Comparison of Training & Testing Time (in seconds).*

Methods	RFC	CNN	TGRL	SEB	SEB/RFC	SEB/CNN	SEB/TGRL
Training	4430	10396	30019	1767	2.6x	5.8x	17.4x
Testing	1284	559	2014	1339	2.3x	3.6x	3.6x
Total	**5714**	**11735**	**31033**	**2326**	**2.5x**	**5.1x**	**13.4x**

Clearly, our approach provides the best efficiency across benchmarks. TGRL lags far behind the others in time efficiency due to the utilization of reinforcement learning that requires tremendous debugging work and parameter tuning work. While CNN provides better time efficiency than TGRL, the two RF-based approaches (RFC and SEB) are significantly faster, and our proposed method is even 2.5x faster than RFC. There are two major advantages of SEB over RFC. First, RFC dumps all data samples and collected features into training phase, which cost extra computation time for handling redundant features and duplicated samples, while SEB perform partial sampling by Shapley analysis. Second, the tree structure of RFC is more complicated than that in SEB. SEB trains a sequence of lightweight decision trees and generate aggregate prediction. The training time of each tree in SEB is much shorter than that in RFC. Overall, our proposed approach drastically (up to 2.5x, 13.4x, and 5.1x on average) improves the time efficiency compared to state-of-the-art methods.

4.5 Robustness Analysis

As discussed in Section 2, an ML model's robustness against obfuscation is an important consideration. In [8], an adversarial attack against ML-based HT detection methods was proposed. Specifically, the author crafted adversarial samples by introducing tiny changes to the gate-level netlists to mess up the statistical features. The standard way of defending against adversarial attack is through adversarial training, where adversarial samples are added to the training set and retrain the entire model. But retraining can be expensive in terms of time. Moreover, if the adversary crafts new adversarial samples, the model has to be retrained again to enhance its robustness. In our work, we explored the robustness of all considered methods against this state-of-the-art adversarial attacks, shown in Table 5. We list the baseline detection accuracy, accuracy under adversarial attack, accuracy after adversarial training, and the extra time (Ex-Time) for adversarial training.

Table 5: *Comparison of accuracy (%) under adversarial attack & extra time needed for retraining models (in seconds).*

Methods	RFC	CNN	TGRL	SEB	SEB/RFC	SEB/CNN	SEB/TGRL
Baseline	75.3	88.2	93.8	99.9	+24.6%	+11.7%	+6.1%
Adversarial	44.3	33.1	50.6	49.2	+4.9%	+ 16.1%	-1.4%
Retrained	73.3	84.9	95.0	98.0	+24.7%	+14.1%	+3.0%
Ex-Time	1500	2677	10926	108	13.8x	24.7x	101.1x

The accuracy of all models against these adversarial samples encounter significant drop, while retraining significantly improves their performance. SEB's retraining time is significantly faster than the others. For all the other methods, retraining is equivalent to retrain the entire model. This advantage is due to the fact that SEB is an adaptive learning framework. In our framework, the retraining happens by introducing a new decision tree into the framework which specifically targets the adversarial samples. The cost for complete retraining is equivalent to only one extra iteration as discussed in Section 3.4. Therefore, our proposed approach is flexible and adaptive to handle various obfuscation techniques.

5 Conclusion

Detection of hardware Trojans is an emerging and urgent need to address semiconductor supply chain vulnerabilities. While there are promising ML-based techniques, they are not useful in practice due to their inherent fundamental limitations. In this work, a boosting machine model is enhanced using Shapely value analysis to build an effective and robust machine learning model. Features derived by Shapley analysis are used to build the boosting framework. The proposed method made several important contributions. It explored an efficient combination of explainable ML technique to provide a fresh perspective for feature selection for HT detection task. We also developed the entire framework based on boosting scheme to drastically reduce both the normal and adversarial training time. Experimental results demonstrated that our approach can drastically reduce the test generation time (up to 5.1x) while it is able to detect a vast majority of the Trojans (99.9% on average), which is a significant improvement (up to 24.6%) compared to state-of-the-art methods.

References

[1] [n.d.]. ISCAS'89 Sequential Benchmark Circuits. https://filebox.ece.vt.edu/~mhsiao/iscas89.html.

[2] R. Chakraborty et al. 2009. MERO: A Statistical Approach for Hardware Trojan Detection. In *CHES*. 396–410.

[3] Hasegawa et al. 2017. Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier. In *ISCAS*.

[4] Yuanwen Huang et al. 2018. Scalable test generation for Trojan detection using side channel analysis. *IEEE TIFS* 13, 11 (2018), 2746–2760.

[5] Roger J Lewis. 2000. An introduction to classification and regression tree (CART) analysis. In *SAEM*, Vol. 14.

[6] Yangdi Lyu and Prabhat Mishra. 2020. MaxSense: Side-Channel Sensitivity Maximization for Trojan Detection using Statistical Test Patterns. *ACM TODAES* (2020).

[7] Yangdi Lyu and Prabhat Mishra. 2020. Scalable Activation of Rare Triggers in Hardware Trojans by Repeated Maximal Clique Sampling. *IEEE TCAD* (2020).

[8] Nozawa et al. 2021. Generating adversarial examples for hardware-trojan detection at gate-level netlists. *Journal of Information Processing* (2021).

[9] Zhixin Pan et al. 2020. Test Generation using Reinforcement Learning for Delay-based Side-Channel Analysis. ICCAD.

[10] Zhixin Pan and Prabhat Mishra. 2021. Automated test generation for hardware trojan detection using reinforcement learning. In *ASPDAC*.

[11] Alvin Roth. 1988. *Shapley value: essays in honor of Lloyd Shapley*. CMU.

[12] Salmani et al. 2013. On design vulnerability analysis and trust benchmarks development. In *ICCD*.

[13] H. Salmani. 2017. COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist. *TIFS* (2017).

[14] Richa Sharma, Vijaypal Singh Rathor, GK Sharma, and Manisha Pattanaik. 2021. A new hardware Trojan detection technique using deep convolutional neural network. *Integration* 79 (2021), 1–11.