PAPER Special Section on Circuits and Systems

Hardware-Trojan Detection at Gate-Level Netlists Using a Gradient Boosting Decision Tree Model and Its Extension Using Trojan Probability Propagation

Ryotaro NEGISHI[†a)], **Tatsuki KURIHARA**[†], ***Nonmembers***, ***and*** **Nozomu TOGAWA**[†], ***Senior Member***

SUMMARY Technological devices have become deeply embedded in people's lives, and their demand is growing every year. It has been indicated that outsourcing the design and manufacturing of integrated circuits, which are essential for technological devices, may lead to the insertion of malicious circuitry, called hardware Trojans (HTs). This paper proposes an HT detection method at gate-level netlists based on XGBoost, one of the best gradient boosting decision tree models. We first propose the optimal set of HT features among many feature candidates at a netlist level through thorough evaluations. Then, we construct an XGBoost-based HT detection method with its optimized hyperparameters. Evaluation experiments were conducted on the netlists from Trust-HUB benchmarks and showed the average F-measure of 0.842 using the proposed method. Also, we newly propose a Trojan probability propagation method that effectively corrects the HT detection results and apply it to the results obtained by XGBoost-based HT detection. Evaluation experiments showed that the average F-measure is improved to 0.861. This value is 0.194 points higher than that of the existing best method proposed so far.

key words: *hardware Trojan, hardware security, netlist, machine learning, gradient boosting tree, XGBoost*

1. Introduction

Technological devices have become deeply embedded in people's lives, and their demand is growing every year. The design and manufacturing of integrated circuits (ICs), which are essential for those technological devices, becomes a large-scale business, and companies often outsource them. While outsourcing IC design and manufacturing to third parties has the benefit of reducing costs, it is also indicated the risk of inserting hardware Trojans (HTs) due to unreliable vendors involved [1]. An HT is a malicious circuit inserted into hardware, whose behavior includes leaking encrypted information, degrading the performance of the device, destroying the device itself, or modifying its functionality.

Considering the current widespread use of ICs such as in washing machines, transportation systems for trains and airplanes, medical instruments, and military equipment intended for military operations, HTs have the potential to affect our daily lives and cause life-threatening situations. In addition, even if countermeasures against a specific HT are taken, a new HT may be developed to counter them. To reduce the threats posed by such HTs, machine-learning-based HT detection methods have been actively researched, especially for detecting HTs introduced at the design stage [2], [3].

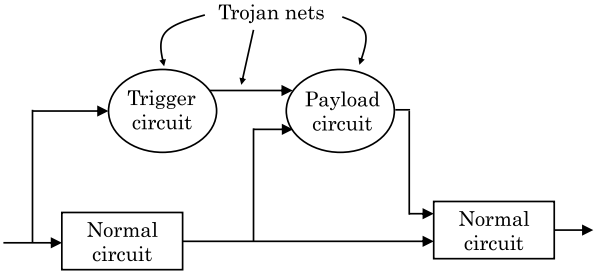


Fig. 1 HT circuit model.

Figure 1 shows the general model of an HT circuit [4]. The trigger circuit generates a trigger signal to activate the HT when the circuit meets certain conditions. The payload circuit uses the trigger signal to activate the HT. The nets constituting an entire circuit can be divided into two classes, Trojan nets and normal nets, depending on whether or not they are included in HTs.

Now, we focus on the design stage of ICs and use machine learning to identify HTs using gate-level netlist features. Detecting HTs at the design stage can save time and cost compared to inspecting all products after manufacturing. Many methods have been proposed for HT detection based on machine learning at a netlist level. In [5], a support vector machine and a neural network are used for detecting HTs, where five HT features for every net are utilized. In [6] and [7], a multi-layer neural network and a random forest model are used, respectively, where 11 HT features for every net are utilized. Among those presented so far, *ensemble learning models* such as a random forest model give good results for HT detection. Although these methods have shown relatively high accuracy in HT detection, they may not always use the best-known learning model for HT detection, and they do not always optimize the netlist features, either.

In this paper, we focus on the gradient boosting decision tree model [8], one of the effective ensemble learning models proposed recently. Particularly, we utilize XGBoost and propose the optimal HT feature set composed of the 24 features for it, among many HT feature candidates. We then propose an HT detection method based on XGBoost with optimized

Manuscript received January 19, 2023.
Manuscript revised May 22, 2023.
Manuscript publicized August 16, 2023.

[†]The authors are with the Department of Computer Science and Communications Engineering, Waseda University, Tokyo, 169-8555 Japan.
a) E-mail: ryotaro.negishi@togawa.cs.waseda.ac.jp

DOI: 10.1587/transfun.2023KEP0005

hyperparameters. The proposed method achieved the average F-measure of 0.842 in the evaluation experiments.

Also, we newly propose a Trojan probability propagation method that effectively corrects the HT detection results. An HT detection method should satisfy the two targets (See Sect. 2 in detail): (T1) it should detect nets that constitute HTs; (T2) it should detect as many HT parts in the netlist as possible. (T1) can be achieved with XGBoost-based HT detection. The proposed Trojan probability propagation method is more capable of satisfying (T2). We apply it to the results obtained by XGBoost-based HT detection.

Evaluation experiments showed that the average F-measure is improved to 0.861. This value is 0.194 points higher than that of the existing best method[†].

The contributions of this paper are summarized as follows:

1. We propose the optimal HT feature set composed of the 24 features for XGBoost by extracting feature importance from the existing 51 HT features and 25 HT features.
2. We optimize the hyperparameters for XGBoost and propose an HT detection method based on hyperparameter-tuned XGBoost.
3. We propose a Trojan probability propagation method that effectively corrects the HT detection results. Hyperparameter-tuned XGBoost with Trojan probability propagation method achieved an average F-measure of 0.861 in HT detection. This is an improvement of 0.194 points compared to the best-known F-measure of 0.667 [10].

The rest of this paper is organized as follows. Section 2 overviews machine-learning-based HT detection and discusses the best models for HT detection. In Sect. 3, we first propose optimal Trojan net features for HT detection using XGBoost, one of the best gradient boosting decision tree models, and then propose an HT detection method based on hyperparameter-tuned XGBoost. After that, we demonstrate its evaluation results. In Sect. 4, we propose a Trojan probability propagation method and apply it to the result of hyperparameter-tuned XGBoost. In Sect. 5, we discuss the validity of our proposed method. Section 6 concludes this paper.

2. HT Detection Using Machine Learning

2.1 Flow of HT Detection

The overall flow of machine learning for HT detection at gate-level netlists is shown in Fig. 2. In HT detection based on machine learning using gate-level netlist features, the learning process extracts HT features for every net in a given netlist in which each net has a label indicating whether it is a normal net or a Trojan net, and a classifier is generated from the extracted features by a machine learning algorithm. Then, the features extracted from the netlist to be tested are input to the classifier generated in the learning process to identify whether each net in the tested netlist is a normal net or a Trojan net.

2.2 Measures for Detection Method

The metrics shown in Table 1 can be used as a measure of the performance of the classifier generated by the machine learning algorithm. In HT detection, TPR (True Positive Rate), which indicates the percentage of Trojan nets that are correctly identified as Trojan nets, and the precision, which indicates the percentage of nets classified as Trojan nets that are truly Trojan nets, are important from the viewpoint of accurately detecting Trojan nets from HT-infected circuits [11]. The F-measure is a metric that shows the balance between the TPR and the precision. Since there is a trade-off relationship between the TPR and the precision, we can evaluate the TPR and the precision in a balanced manner by using the F-measure as an evaluation metric. In this paper, we focus on maximizing the F-measure to evaluate the detection method.

2.3 Machine Learning Models and Features for HT Detection

There have been proposed several HT detection methods using machine learning models, such as in [5]–[7], [10], [13]. In [5], a support vector machine and a neural network are used as machine learning models. The method using a support vector machine achieved the average TPR of 83%, the average TNR (True Negative Rate) of 49%, and the average accuracy of 51%. The neural network-based method achieved the average TPR of 81%, the average TNR of 69%, and the average accuracy of 69%. In [6], a multi-layer neural-network-based method is proposed which achieved the average TPR of 85%, and the average TNR of 70%. In [7], a random-forest-based method is proposed, which achieved the average TPR of 70.3%, the average TNR of 99.7%, and the average accuracy of 99.2%. In [13], a bagged-tree-based method is proposed, which achieved the average TPR of 82.46%, the average TNR of 98.99%, and the average accuracy of 98.26%. These methods achieved relatively high classification results but they all used a limited set of benchmarks from Trust-HUB [12]. In [10], a random-forest-based

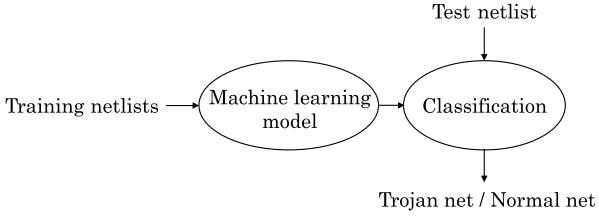


Fig. 2 Machine learning flow for HT detection.

[†]The preliminary version of this paper appeared in [9]. The main extensions are summarized as follows: We propose a Trojan probability propagation method and its evaluation results in Sect. 4. Furthermore, we deepen the explanation and discussion in this paper.

Table 1 Definition of metrics.

Metrics	Definition
TP	The number of Trojan nets correctly classified as Trojan nets.
TN	The number of normal nets correctly classified as normal nets.
FP	The number of normal nets incorrectly classified as Trojan nets.
FN	The number of Trojan nets incorrectly classified as normal nets.
TPR	The percentage of Trojan nets that are correctly classified as Trojan nets, which is defined by TP/(TP + FN)
TNR	The percentage of normal nets that are correctly classified as normal nets, which is defined by TN/(TN + FP)
Precision	The percentage of nets that are truly Trojan nets out of those determined to be Trojan nets, which is defined by TP/(TP + FP)
F-measure	Harmonic mean of TPR and Precision, which is defined by 2/(1/TPR + 1/Precision)
Accuracy	The percentage of all nets that are correctly classified, which is defined by (TP + TN)/(TP + TN + FP + FN)

Table 2 The HT features for HT detection ($1 \leq x, y \leq 5$) [7], [10].

Feature	Description
fan_in_x	The number of logic-gate fanins up to x-level away from the input side of the net.
in_flipflop_x	The number of flip-flops up to x-level away from the input side of the net.
out_flipflop_x	The number of flip-flops up to x-level away from the output side of the net.
in_multiplexer_x	The number of multiplexers up to x-level away from the input side of the net.
out_multiplexer_x	The number of multiplexers up to x-level away from the output side of the net.
in_loop_x	The number of up to x-level loops on the input side.
out_loop_x	The number of up to x-level loops on the input side.
in_const_x	The number of constants up to x-level away from the input side of the net.
out_const_x	The number of constants up to x-level away from the output side of the net.
in_nearest_pin	The minimum level to the primary input from the net.
out_nearest_pout	The minimum level to the primary output from the net.
{in, out}_nearest_flipflop	The minimum level to any flip-flop from the input or output side of the net.
{in, out}_nearest_multiplexer	The minimum level to any multiplexer from the input or output side of the net.
fan_in_uxdy	Starting from the target net n, we firstly go down by x gates to the output side. Then we go up by y gates to the input side and count the number of all the fanins there. This count gives fan_in_uxdy for the net n.

Table 3 Netlists from Trust-HUB [12].

Netlist	# of normal nets	# of Trojan nets
RS232-T1000	309	10
RS232-T1100	309	11
RS232-T1200	310	13
RS232-T1300	309	7
RS232-T1400	306	12
RS232-T1500	311	11
RS232-T1600	311	10
s15850-T100	2,420	26
s35932-T100	6,408	14
s35932-T200	6,405	12
s35932-T300	6,405	37
s38417-T100	5,799	11
s38417-T200	5,802	11
s38417-T300	5,801	44
s38584-T100	7,343	19
s38584-T200	7,373	97
s38584-T300	7,615	873
EthernetMAC10GE-T700	102,969	12
EthernetMAC10GE-T710	102,969	12
EthernetMAC10GE-T720	102,969	12
EthernetMAC10GE-T730	102,969	12
B19-T100	70,649	96
B19-T200	70,649	96
wb_conmax-T100	22,186	11
RS232-free	303	0
s15850-free	2,419	0
s35932-free	6,405	0
s38417-free	5,798	0
s38584-free	7,343	0
EthernetMAC10GE-free	102,967	0
B19-free	70,618	0
wb_conmax-free	22,182	0

method is also proposed using the different HT feature sets for the netlists in Table 2, which achieved the average TPR of 63.6%, and the average TNR of 100.0%. This method [10] evaluated various netlists with a maximum of 100K nets and achieved the best classification results. As far as we know, the method [10] realized the best F-measure for the netlists in Table 3.

From the above results, we can see that an *ensemble learning model* such as the random forest and the bagged tree is particularly effective in HT detection. This is mainly because, many features must relate to identifying HTs, such as the number of fan-ins to every gate and the distance to flip-flops, but we cannot say which one of them definitely determines HTs. In such cases, ensemble learning models can effectively identify HTs by using various decision trees based on HT features.

Furthermore, many HT features for a signal net have been proposed to identify HTs [7], [10]. In [7], a total of 51 HT-related features are proposed. In [10], additional 25 HT features are proposed focusing on the structure of trigger circuits (See Table 2. The details will be discussed in Sect. 3). These features have been evaluated under different conditions, and the optimal ones for ensemble learning models have not been discussed so far.

Recently, a gradient boosting decision tree model has been proposed as an ensemble learning model, which generally achieves high accuracy in classification [8]. The gradient boosting decision tree model is a machine learning model that combines a gradient descent method, ensemble learning, and decision trees. Ensemble learning is a model that improves prediction performance by using multiple simple weak classifiers to build a linear combination. In the gradient boosting decision tree model, decision trees are used

as weak learners for boosting. XGBoost is one of the best gradient boosting decision tree models [8], which must be well applied to HT detection due to the previous discussions.

When we use an HT detection method at gate-level netlists, we want to detect HT included in the netlist. Then an HT detection method should satisfy the two targets below:

(T1) An HT detection method should detect nets that constitute HTs.
(T2) An HT detection method should detect as many HT parts in the netlist as possible.

(T1) can be achieved with machine-learning-based HT detection. In this paper, we achieve it with the XGBoost-based method. However, we do not consider that we can use all the circuit information related to hardware Trojans in the machine learning phase. There must still remain circuit information that we can take into account after the machine-learning-based HT detection phase (e.g., whether a nearby net is Trojan or normal). Finding more Trojan nets using machine-learning-based HT detection results together with additional circuit information can more satisfy the target (T2). Therefore, we propose a Trojan probability propagation method to satisfy the target (T2).

Hereafter, this paper assumes the use of XGBoost for HT detection and aims at optimizing the HT features for every signal net and the structure of the gradient boosting decision trees for this purpose. Furthermore, we make the correction method, called a Trojan probability propagation method, available to the machine learning results, allowing more HT locations to be detected.

3. HT Detection Using XGBoost

In this section, we propose an HT detection method using XGBoost. First, we propose the optimal set of the HT features for a signal net, referring to the HT features proposed so far (Sect. 3.1). Second, we perform the detailed hyperparameter optimization on XGBoost (Sect. 3.2). Finally, we construct the proposed hyperparameter-tuned XGBoost-based method and show the results of HT detection comparing to the existing best results (Sect. 3.3).

The evaluation environment for this section is summarized as follows. A computer with 1.0TB of memory, Intel Xeon Platinum 8180M, Python 3.9.2 and xgboost 1.4.2 were used. The netlists to be evaluated were the 32 different netlists from Trust-HUB [12], as shown in Table 3. The first 24 netlists are Trojan netlists with HTs inserted, and the remaining eight netlists are normal netlists with no HTs inserted.

According to [9], we evaluate the machine learning model by leave-one-out cross validation. In other words, when evaluating a netlist N in Table 3, we train the machine learning model on the remaining 31 netlists, and then for each net in N, we identify whether the net is a Trojan net or a normal net. For each netlist, the metrics in Table 1 are calculated, and the average of the metrics over the 32 netlists is used as the evaluation metrics. For each validation, the training data are compared, and if there are nets with the same feature values, the nets are excluded from the training data. The threshold for classification is set to 0.5.

3.1 Feature Importance Extraction and HT Feature Set Optimization

Many HT features for a signal net n have been proposed by [7] and [10]. Table 2 summarizes them. In Table 2, the 51 features from fan_in_x to {in, out}_nearest_multiplexer are the features proposed in [7], and the 25 features of fan_in_uxdy are the features proposed in [10][†].

We now use all the HT features proposed in [7], [10]. The function included in the XGBoost library was used to extract the feature importance (See Appendix in detail). Here, the importance refers to the degree to which the node in the decision tree containing the feature contributes to the improvement of the objective function in the entire model when training. In this evaluation, we used the default values [14] of the hyperparameters from the XGBoost library. This is because it is difficult to extract a feature set with varying hyperparameter values and it is uncertain how to setup the hyperparameter values at this phase. We performed the leave-one-out cross validation for each netlist in Table 3 and obtained the importance for every feature in Table 2. We averaged the importance for every feature over all the netlists and obtained the importance values.

Table 4 shows the results of the evaluations. Among the 76 features, fan_in_u4d1 and fan_in_u5d1 have particularly high importance. This is because, the trigger circuits in HTs have the structure as in Fig. 3, which is included in s15850-T100 from Table 3. When we focus on the net Tg1_OUT1 in the HT in Fig. 3(a), its fan_in_u4d1 value becomes two, which is relatively small compared to normal nets. The trigger circuit often has the pyramidal structure in which the large fan-ins make a trigger condition and its signal is input to the payload circuit. Then the value of fan_in_u4d1 tends to become small. In a similar way, Fig. 3(b) shows an example of fan_in_u5d1, which also becomes small. These values well characterize the HT structure and thus its importance becomes large in Table 4. In addition, out_const_x, in_loop_1, and out_loop_1 have no importance, indicating that they are not useful for HT detection. This is because, HTs do not include short-length loop structures nor constant inputs and hence these values do not contribute to identifying HTs at all.

Next, the optimal number of features is measured using the extracted importance. We extract the top k HT features from Table 4 and classify all the nets in Table 3. Then, we measure the five metrics of Table 1. Figure 4 summarizes the results. All the five metrics decrease sharply when $k < 25$. The average F-measure is highest when 54 features

[†]In [7], the number of features was narrowed down to 11 out of 51 based on the feature importance. However, Ref. [7] only discusses the importance of the 51 features for 16 netlists, and no evaluation of importance has been made for the entire 76 features including those proposed in [10].

Table 4 Feature importance.

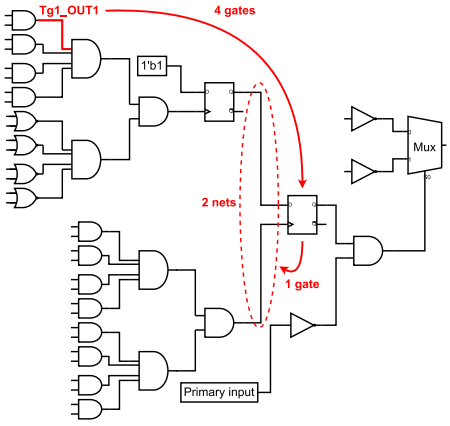
#	Feature	Importance	#	Feature	Importance
1	fan_in_u4d1	8146.61	39	out_multiplexer_1	195.91
2	fan_in_u5d1	6714.52	40	fan_in_u5d4	185.80
3	fan_in_u5d5	2480.82	41	in_flipflop_5	170.64
4	out_nearest_pout	2292.50	42	fan_in_u3d4	121.67
5	out_nearest_multiplexer	1625.44	43	out_multiplexer_4	121.48
6	out_flipflop_3	1479.41	44	fan_in_u1d2	107.25
7	out_nearest_flipflop	1278.23	45	fan_in_u4d5	104.90
8	fan_in_u3d5	786.76	46	fan_in_5	102.50
9	fan_in_u1d3	775.88	47	fan_in_u3d3	100.58
10	fan_in_u2d3	751.77	48	out_flipflop_1	91.99
11	fan_in_u2d5	733.82	49	out_multiplexer_3	91.33
12	in_nearest_flipflop	705.91	50	fan_in_u3d1	79.24
13	out_flipflop_5	620.97	51	in_flipflop_4	63.08
14	fan_in_u1d4	602.15	52	out_loop_3	56.81
15	fan_in_u2d1	584.88	53	fan_in_1	48.32
16	fan_in_u2d2	567.07	54	in_loop_3	47.63
17	in_const_4	565.62	55	out_loop_5	43.54
18	fan_in_3	540.46	56	fan_in_u2d4	36.49
19	in_loop_2	506.67	57	out_multiplexer_2	30.16
20	fan_in_u4d2	503.01	58	fan_in_4	28.99
21	fan_in_u1d5	433.83	59	in_const_3	23.52
22	fan_in_u1d1	409.01	60	in_flipflop_1	22.77
23	out_flipflop_4	392.62	61	in_loop_5	18.53
24	out_multiplexer_5	390.86	62	out_loop_4	14.69
25	in_multiplexer_5	374.56	63	in_const_1	2.15
26	in_loop_4	372.87	64	in_multiplexer_2	1.71
27	in_nearest_pin	354.28	65	in_const_2	1.51
28	fan_in_u3d2	346.16	66	in_multiplexer_3	0.76
29	in_flipflop_3	324.25	67	out_loop_2	0.73
30	fan_in_u5d2	303.86	68	in_multiplexer_1	0.37
31	fan_in_u5d3	273.86	69	in_multiplexer_4	0.29
32	fan_in_u4d4	265.36	70	in_loop_1	0
33	in_flipflop_2	241.87	71	out_loop_1	0
34	fan_in_2	233.05	72	out_const_1	0
35	in_const_5	219.22	73	out_const_2	0
36	out_flipflop_2	216.16	74	out_const_3	0
37	in_nearest_multiplexer	212.79	75	out_const_4	0
38	fan_in_u4d3	211.87	76	out_const_5	0

are used, reaching 0.601. However, the average F-measure also peaks at 0.592 when the 28 features are used, which is almost the same as the highest F-measure value. In general, using too many features in machine learning causes overfitting, which may result in degrading the classification performance. Hence, the 28 HT features from #1–#28 in Table 4 are determined to be optimal for XGBoost†.

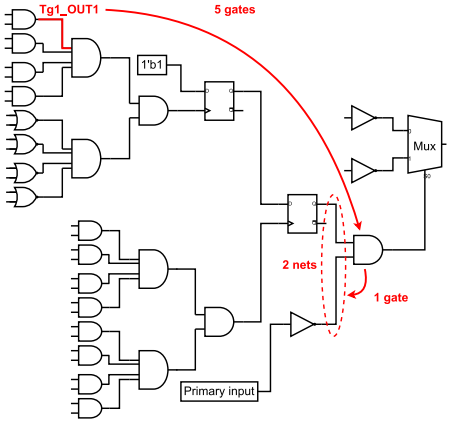
3.2 XGBoost Hyperparameter Tuning

We tune the hyperparameters in XGBoost in detail. Firstly, we focus on the hyperparameters of *early_stopping_rounds*, *num_boost_round*, *objective*, and *scale_post_weight*. When the validation error does not decrease at least every *early_stopping_rounds* iterations, the model stops learning to avoid overlearning. The default value of *early_stopping_rounds* is 10. However, If we set *early_stopping_rounds* to a small value, the model is likely to stop learning due to a temporary increase in losses. Hence, we set it to 20. *num_boost_round* is set to a large number, 2000, so that early stopping works at almost every validation. For HT detection at gate-level netlists, we want to see whether the net is a normal net or a Trojan net, and hence we set the *objective* to reg:logistic, i.e., logistic regression.

†If we select 54 features from #1 to #54 in Table 4, we cannot achieve the F-measure of 0.842 discussed in Sect. 3.3, even after we optimize the hyperparameters. We consider that this is because overfitting occurs due to using too many features.



(a) The fan_in_u4d1 value for the net Tg1_OUT1 (fan_in_u4d1 = 2).



(b) The fan_in_u5d1 value for the net Tg1_OUT1 (fan_in_u5d1 = 2).

Fig. 3 HT in s15850-T100.

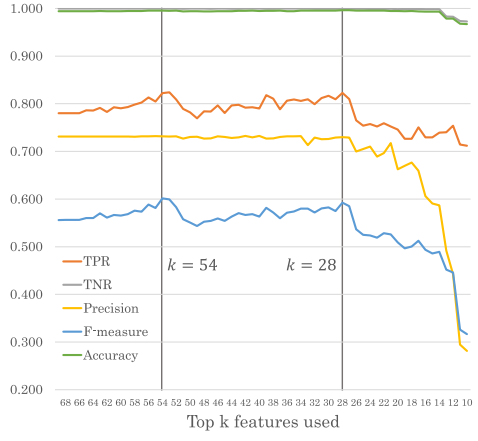


Fig. 4 The relationship between the number of features used and the five metrics of the classifier.

In most HT netlists, the percentage of Trojan nets is much small compared to the number of total nets. This causes an imbalance between the number of normal nets and the number of Trojan nets in the training data, which leads to inaccurate learning. Setting *scale_pos_weight* to the ratio of the number of normal nets to the number of Trojan nets in

Table 5 Hyperparameters in XGBoost (Part 1).

Hyperparameter	Value
num_boost_round	2000
early_stopping_rounds	20
objective	reg:logistic
scale_pos_weight	ratio of # of normal nets to # of Trojan nets

Table 6 Hyperparameters in XGBoost (Part 2).

Hyperparameter	Candidate values	Optimal value
eta	[0.05, 0.10, 0.15, 0.20]	0.10
max_depth	[4, 5, 6]	4
min_child_weight	[1, 2, 3, 4, 5, 6]	1
gamma	[0, 0.2, 0.4, 0.6, 0.8, 1.0]	0.2
subsample	[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	1.0
colsample_bytree	[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	0.8

the training data corrects the balance of positive and negative weights. Table 5 summarizes the hyperparameter setting in XGBoost here.

After that, we optimize the hyperparameters thoroughly enumerated in Table 6 by using a random search [15]. When applying a random search, the cross validation using the 32 Trust-HUB netlists is performed and the F-measure is maximized. The results are summarized in the rightmost column of Table 6. The other hyperparameters are set to the default values. By optimizing the hyperparameters, XGBoost achieves the F-measure of 0.842 as discussed in the next subsection.

3.3 Proposed Method and Its Evaluations

Based on the above discussion, the proposed HT detection method is constructed such that we utilize XGBoost as a machine learning model with the hyperparameters of Table 5 and Table 6, using the 28 HT features in Table 4. Then we evaluate the proposed method. Table 7 shows the results of the proposed method and those in [10] for comparison.

When comparing the proposed method to [10], the average TPR of the proposed method is 0.188 points higher and its average precision is 0.025 points lower. This is because the proposed method identifies slightly more normal nets as Trojan nets than the existing method [10], but successfully detects more Trojan nets. In terms of the average F-measure, the proposed method achieves 0.175 points higher, indicating that the proposed method is superior when the average TPR and the average precision are considered together.

Table 10 and Table 11 show the detailed results of the proposed method and those of [10], respectively. In terms of TPR, the proposed method shows higher values for those netlists, such as s38584 series and wb_conmax-T100, where the TPR values of these netlists are significantly lower than the other netlists when using the existing method [10]. In particular, the TPR value of s38417-T100 and wb_conmax-T100 is improved by 0.727 points. In terms of precision, the error of identifying normal nets as Trojan nets in normal netlists by the proposed method is remarkably low. In the HT netlists such as RS232-T1600 and s38584-T100, up to 0.189-point improvement is seen in the precision. In terms of F-measure, there is an overall improvement. While the F-measure of RS232-T1000 of the proposed method is decreased by just 0.048 points, other netlists such as s35932-T100 and s38417-T100 achieve the F-measure improvement of 0.700 points or more. Overall, the proposed method achieves the significantly higher F-measure than the existing best method [10].

Table 7 Comparison of the proposed method and the existing method [10].

Method	TPR	TNR	Precision	F-measure	Accuracy
Ours (tuned XGBoost)	0.824	0.999	0.932	**0.842**	0.996
[10]	0.636	1.000	0.957	0.667	0.994

Note that, we require approximately 20 minutes to construct the XGBoost model using 31 netlists and 0.1 seconds to classify between Trojan nets and normal nets in every netlist in Table 3.

4. Trojan Probability Propagation Method for HT Detection based on Machine Learning

4.1 Trojan Probability Propagation Method

HT detection methods based on machine learning independently identify each net whether it is a Trojan net or a normal net. However, the identification of each net cannot take into account the identification information surrounding the net. In general, HTs exist in a specific location in the netlist. Therefore, we expect to improve the identification performance by applying a correction method to the identification results of HT obtained by machine learning.

We newly propose a Trojan probability propagation method to improve the identification performance by propagating the identification information to re-identify Trojan nets. After applying a machine-learning-based HT detection method, every net has a Trojan probability, which indicates how likely the net is a Trojan net (See Appendix on how to calculate the probability). As in Sect. 3, if the Trojan probability is equal to or larger than the threshold value, we identify it to be the Trojan net. The algorithm of the Trojan probability propagation method and the conditions under which the identification results are propagated are described as follows:

Step 1. Obtain the Trojan probability of each net using an After applying a machine-learning-based HT detection method based on machine learning.

Step 2. Identify the nets with Trojan probability equal to or larger than the threshold α_0 (> 0) as Trojan nets. α_0 is set to 0.5 as in Sect. 3.

Step 3. Traverse the netlist from the input side to the output side and check if either one of the following two conditions is satisfied. If so, re-identify the net to be the Trojan net.

(Condition 1) At a gate i, if all the nets on its input (output) side are identified as Trojan nets, then all the nets on the output (input) side of the gate i are

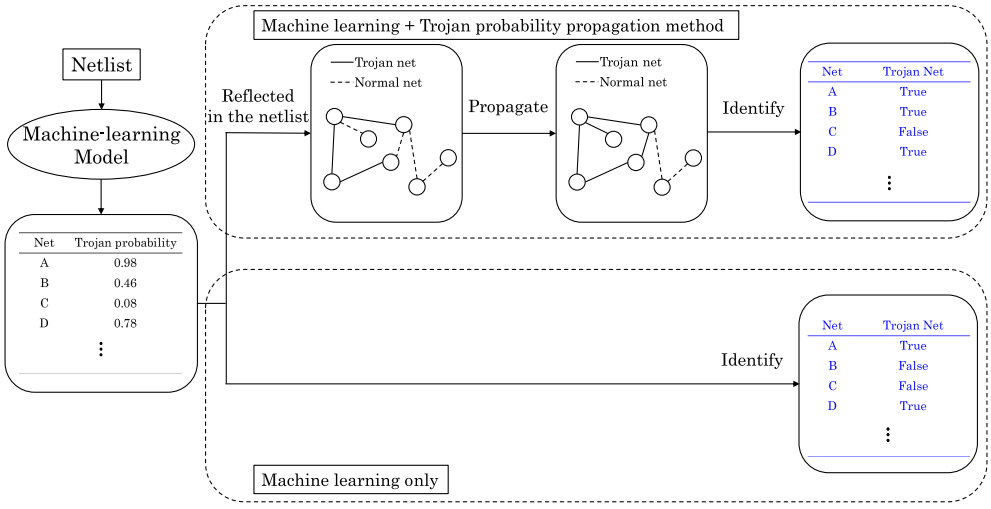


Fig. 5 Comparison of the behavior whether or not the Trojan probability propagation method is applied.

identified as Trojan nets.

(Condition 2) At a net n identified as a normal net, if n is connected to a net identified as a Trojan net and the Trojan probability of n is equal to or greater than the threshold α_1 ($> \alpha_0$), then we identify n as the Trojan net.

Step 4. Repeat Step 3 until all the nets do not satisfy the propagation conditions above.

Condition 1 shows that HTs exist locally in every netlist and hence, if all the nets on the input (output) side of a gate are Trojan nets, we can consider that all the nets on its output (input) side are also Trojan nets. **Condition 2** also shows that HTs exist locally in every netlist and hence, when focusing on a normal net n with relatively high Trojan probability, it is changed to a Trojan net if there exists a Trojan net neighboring to it.

Figure 5 shows the behavior of "machine learning + Trojan probability propagation method" and "machine learning" only.

4.2 Evaluations of Trojan Probability Propagation Method

We applied the Trojan probability propagation method to the results from the hyperparameter-tuned XGBoost in Sect. 4.2. The threshold α_1 of Trojan probability propagation method is 0.4[†]. Table 12 shows the evaluation results. By applying the Trojan probability propagation method, an average TPR of 0.890, an average precision of 0.880, and an average F-measure of 0.861 were achieved. When comparing before and after applying the Trojan probability propagation method, the average precision slightly decreased by 0.052 points, but the average TPR increased by 0.066 points, resulting in a 0.019 point increase in the average F-measure. Furthermore, the TPR for each benchmark remained unchanged or improved, while the TNR almost remained unchanged. This is because the Trojan probability propagation method only propagates Trojan nets, so that what is identified as a normal net is re-identified as a Trojan net. When comparing to [10], the average F-measure increased by 0.194 points.

Note that the Trojan probability propagation method requires approximately 20 seconds in each netlist of Table 3.

[†]We tried various α_1 values and obtained the best results when α_1 is set to 0.4.

5. Discussion

5.1 Random Search

In this paper, we use random search to optimize the XGBoost model for HT detection in Sect. 3.2. This is due to the following reason.

In general, grid search and random search are often used for hyperparameter tuning in machine learning models [15]. Grid search is a method that searches for all combinations of candidate hyperparameter values. Grid search has the advantage of always finding the optimal solution. Random search is a method that randomly searches for a combination of candidate hyperparameter values with an upper limit on the number of times that the search can be performed and it finds reasonably good hyperparameters.

In [15], Bergstra et al. found by computational experiments that only a small fraction of the total hyperparameters of a machine learning model are important with respect to performance. They call such a property *low effective dimensionality* (LED, in short). An important advantage of random search is its robustness to LED compared to grid search. Grid search searches for all combinations, regardless of the impact of a hyperparameter on model performance. On the other hand, random search always determines the values of all hyperparameters randomly. Thus, random search does not tend to produce useless objective function evaluations due to the presence of ineffective hyperparameters.

As described above, random search has the advantage

over grid search. Further, by using random search, we can optimize the XGBoost model so that we can achieve the F-measure that outperforms the previous results as in Table 7. We consider that using random search is a good option to optimize XGBoost for HT detection.

5.2 Hyperparameters of XGBoost

In Sect. 3.2, we tuned the hyperparameters in Table 6 with the hyperparameters in Table 5 fixed. This is because of the following reasons:

First, we discuss why we tuned the hyperparameters in Table 6. XGBoost has many hyperparameters that can be set by the user. However, as discussed in Sect. 5.1, many machine learning hyperparameters cannot affect model performance very much. For example, an XGBoost-based HT detection method [18] trains the model with its own hyperparameters including *eta*, *max_depth*, and *min_child_weight*. Further, in [19], [20], the XGBoost models are tuned under the hyperparameters of *gamma*, *subsample*, and *colsample_bytree*, though their target does not HT detection. Hence, we tuned in our XGBoost model the hyperparameters of *eta*, *max_depth*, *min_child_weight*, *gamma*, *subsample*, and *col-*

Table 8 The result when setting the default values to the hyperparameters in Table 5.

Method	TPR	TNR	Precision	F-measure	Accuracy
Default tuned XGBoost	0.663	1.000	0.990	0.775	0.995

Table 9 Comparison of the proposed method and existing XGBoost-based methods [16]–[18].

Method	# of Netlists	TPR	TNR	Precision	F-measure	Accuracy
Ours (tuned XGBoost)	32	0.824	0.999	0.932	0.842	0.996
Ours (XGBoost + Propagation Method)	32	0.890	0.997	0.880	0.861	0.996
Ours (tuned XGBoost)	16*	0.768	0.998	0.931	0.796	0.993
Ours (XGBoost + Propagation Method)	16*	0.849	0.995	0.861	0.818	0.992
Ours (tuned XGBoost)	12*	0.819	0.999	0.970	0.843	0.999
Ours (XGBoost + Propagation Method)	12*	0.907	0.996	0.904	0.867	0.996
[16]	16**	0.990	0.990	0.989	0.988	0.990
[17]	12**	0.898	0.999	0.923	0.878	0.998
[18]	11**	0.940	0.993	0.800	0.849	0.990

* The results of the 12 netlists or 16 netlists are extracted from Table 10 and Table 12 and summarized.
** All of these netlists are included in the netlists in Table 3.

Table 10 The detailed results of the proposed method (tuned XGBoost).

Netlist	TP	TN	FP	FN	TPR	TNR	Precision	F-measure	Accuracy
RS232-T1000	10	308	1	0	1.000	0.997	0.909	0.952	0.997
RS232-T1100	11	309	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1200	13	310	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1300	7	309	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1400	12	306	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1500	11	309	2	0	1.000	0.994	0.846	0.917	0.994
RS232-T1600	8	311	1	1	0.889	0.997	0.889	0.889	0.994
s15850-T100	18	2420	0	8	0.692	1.000	1.000	0.818	0.997
s35932-T100	11	6409	0	2	0.846	1.000	1.000	0.917	1.000
s35932-T200	1	6405	0	11	0.083	1.000	1.000	0.154	0.998
s35932-T300	34	6405	0	3	0.919	1.000	1.000	0.958	1.000
s38417-T100	9	5799	0	2	0.818	1.000	1.000	0.900	1.000
s38417-T200	3	5802	0	8	0.273	1.000	1.000	0.429	0.999
s38417-T300	44	5800	1	0	1.000	1.000	0.978	0.989	1.000
s38584-T100	3	7343	1	15	0.167	1.000	0.750	0.273	0.998
s38584-T200	104	7304	40	22	0.825	0.995	0.722	0.770	0.992
s38584-T300	540	7212	133	603	0.472	0.982	0.802	0.595	0.913
EthernetMAC10GE-T700	12	102968	1	0	1.000	1.000	0.923	0.960	1.000
EthernetMAC10GE-T710	12	102968	1	0	1.000	1.000	0.923	0.960	1.000
EthernetMAC10GE-T720	12	102968	1	0	1.000	1.000	0.923	0.960	1.000
EthernetMAC10GE-T730	12	102967	2	0	1.000	1.000	0.857	0.923	1.000
B19-T100	95	70645	4	1	0.990	1.000	0.960	0.974	1.000
B19-T200	95	70645	4	1	0.990	1.000	0.960	0.974	1.000
wb_conmax-T100	9	22186	0	2	0.818	1.000	1.000	0.900	1.000
B19-free	0	70612	6	0	-	1.000	-	-	1.000
RS232-free	0	303	0	0	-	1.000	-	-	1.000
s15850-free	0	2419	0	0	-	1.000	-	-	1.000
s35932-free	0	6405	0	0	-	1.000	-	-	1.000
s38417-free	0	5797	1	0	-	1.000	-	-	1.000
s38584-free	0	7341	2	0	-	1.000	-	-	1.000
wb_conmax-free	0	22182	0	0	-	1.000	-	-	1.000
EthernetMAC10GE-free	0	102966	1	0	-	1.000	-	-	1.000
Average	-	-	-	-	0.824	0.999	0.932	0.842	0.996

Table 11 The detailed results of the existing method [10].

Netlist	TP	TN	FP	FN	TPR	TNR	Precision	F-measure	Accuracy
RS232-T1000	10	309	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1100	11	309	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1200	13	310	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1300	6	309	0	1	0.857	1.000	1.000	0.923	0.997
RS232-T1400	12	306	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1500	11	310	1	0	1.000	0.997	0.917	0.957	0.997
RS232-T1600	7	309	3	2	0.778	0.990	0.700	0.737	0.984
s15850-T100	2	2420	0	24	0.077	1.000	1.000	0.143	0.990
s35932-T100	1	6409	0	12	0.077	1.000	1.000	0.143	0.998
s35932-T200	1	6405	0	11	0.083	1.000	1.000	0.154	0.998
s35932-T300	34	6405	0	3	0.919	1.000	1.000	0.958	1.000
s38417-T100	1	5799	0	10	0.091	1.000	1.000	0.167	0.998
s38417-T200	1	5802	0	10	0.091	1.000	1.000	0.167	0.998
s38417-T300	44	5798	3	0	1.000	0.999	0.936	0.967	0.999
s38584-T100	3	7342	2	15	0.167	1.000	0.600	0.261	0.998
s38584-T200	22	7343	1	104	0.175	1.000	0.957	0.295	0.986
s38584-T300	37	7339	6	1106	0.032	0.999	0.860	0.062	0.869
EthernetMAC10GE-T700	12	102969	0	0	1.000	1.000	1.000	1.000	1.000
EthernetMAC10GE-T710	12	102969	0	0	1.000	1.000	1.000	1.000	1.000
EthernetMAC10GE-T720	12	102969	0	0	1.000	1.000	1.000	1.000	1.000
EthernetMAC10GE-T730	10	102969	0	2	0.833	1.000	1.000	0.909	1.000
B19-T100	96	70649	0	0	1.000	1.000	1.000	1.000	1.000
B19-T200	96	70649	0	0	1.000	1.000	1.000	1.000	1.000
wb_conmax-T100	1	22186	0	10	0.091	1.000	1.000	0.167	1.000
B19-free	0	70618	0	0	-	1.000	-	-	1.000
RS232-free	0	303	0	0	-	1.000	-	-	1.000
s15850-free	0	2419	0	0	-	1.000	-	-	1.000
s35932-free	0	6405	0	0	-	1.000	-	-	1.000
s38417-free	0	5798	0	0	-	1.000	-	-	1.000
s38584-free	0	7338	5	0	-	0.999	-	-	0.999
wb_conmax-free	0	22182	0	0	-	1.000	-	-	1.000
EthernetMAC10GE-free	0	102967	0	0	-	1.000	-	-	1.000
Average	-	-	-	-	0.636	1.000	0.957	0.667	0.994

sample_bytree as in Table 6.

Second, we discuss why we fixed the hyperparameter values as in Table 5. A rough reason is shown in Sect. 3.2, but we have also done several preliminary experiments and determined the values as shown in Table 5. For example, assume that we set the default values to all the hyperparameters in Table 5 (the default values are shown in [14]). After that, the hyperparameters in Table 6 are tuned using the random search as discussed in Sect. 3.2. Then the results are obtained as shown in Table 8. Table 8 shows that the F-measure is 0.775, which is 0.067 points less than the model tuned in Sect. 3.2. The hyperparameter values shown in Table 5 are the ones that give the best performance in our preliminary experiments and that is why we fix the hyperparameter values as shown in Table 5.

5.3 Comparison of the Proposed Method and Existing XGBoost-Based Methods

There have been several XGBoost-based HT detection methods proposed [16]–[18]. We compare and evaluate our proposed method with these methods. Table 9 shows the results of the proposed method and those in [16]–[18] for comparison. Note that, some of the metrics are not available in the original references, and then they were derived using the other metrics and summarized in Table 9.

First, we compare the proposed method with the method [16]. In [16], 16 netlists were used for evaluation, all of which are included in the netlists in Table 3. However, all netlists used for evaluation were small to medium size, and the method [16] was not evaluated for the netlists with more than 100K nets. Gate-level IP cores may include 100K or more nets and it must be required to evaluate such large-sized netlists in HT detection practically. On the contrary, the proposed method has been evaluated including the netlists exceeding 100K nets, such as EthernetMAC10GE, and even on HT-free netlists. It is uncertain if the method [16] is applied to such large-sized netlists, but our proposed method outperforms the method [16] in TNR at least. When we extract the results for the 16 netlists used in [16] from Table 10 and Table 12, the results are summarized in the 4th and 5th rows of Table 9. Compared on the same netlists, both the XGBoost-only method and the XGBoost + Trojan probability propagation method outperform the method [16] in TNR and accuracy.

Next, we compare the proposed method with the method [17]. In [17], 12 netlists were used for evaluation, all of which are included in the netlists in Table 3. This method does not evaluate netlists containing more than 100K nets, either. Our proposed method (2nd row) outperforms the method [17] in precision at least. When we extract the results for the 12 netlists used in [17] from Table 10 and Table 12, the results are summarized in the 6th and 7th rows of Table 9. Compared on the same netlists, the XGBoost-only method

Table 12 The detailed results of the Trojan probability propagation method after applying XGBoost-based HT detection.

Netlist	TP	TN	FP	FN	TPR	TNR	Precision	Fmeasure	Accuracy
RS232-T1000	10	307	2	0	1.000	0.994	0.833	0.909	0.994
RS232-T1100	11	307	2	0	1.000	0.994	0.846	0.917	0.994
RS232-T1200	13	306	4	0	1.000	0.987	0.765	0.867	0.988
RS232-T1300	7	309	0	0	1.000	1.000	1.000	1.000	1.000
RS232-T1400	12	305	1	0	1.000	0.997	0.923	0.960	0.997
RS232-T1500	11	307	4	0	1.000	0.987	0.733	0.846	0.988
RS232-T1600	9	309	3	0	1.000	0.990	0.750	0.857	0.991
s15850-T100	24	2420	0	2	0.923	1.000	1.000	0.960	0.999
s35932-T100	12	6409	0	1	0.923	1.000	1.000	0.960	1.000
s35932-T200	1	6405	0	11	0.083	1.000	1.000	0.154	0.998
s35932-T300	36	6405	0	1	0.973	1.000	1.000	0.986	1.000
s38417-T100	11	5799	0	0	1.000	1.000	1.000	1.000	1.000
s38417-T200	10	5802	0	1	0.909	1.000	1.000	0.952	1.000
s38417-T300	44	5796	5	0	1.000	0.999	0.898	0.946	0.999
s38584-T100	3	7341	3	15	0.167	1.000	0.500	0.250	0.998
s38584-T200	104	7301	43	22	0.825	0.994	0.707	0.762	0.991
s38584-T300	747	7186	159	396	0.654	0.978	0.825	0.729	0.935
EthernetMAC10GE-T700	12	102968	1	0	1.000	1.000	0.923	0.960	1.000
EthernetMAC10GE-T710	12	102968	1	0	1.000	1.000	0.923	0.960	1.000
EthernetMAC10GE-T720	12	102967	2	0	1.000	1.000	0.857	0.923	1.000
EthernetMAC10GE-T730	12	102966	3	0	1.000	1.000	0.800	0.889	1.000
B19-T100	96	70641	8	0	1.000	1.000	0.923	0.960	1.000
B19-T200	96	70641	8	0	1.000	1.000	0.923	0.960	1.000
wb_conmax-T100	10	22186	0	1	0.909	1.000	1.000	0.952	1.000
B19-free	0	70604	14	0	-	1.000	-	-	1.000
RS232-free	0	303	0	0	-	1.000	-	-	1.000
s15850-free	0	2419	0	0	-	1.000	-	-	1.000
s35932-free	0	6405	0	0	-	1.000	-	-	1.000
s38417-free	0	5796	2	0	-	1.000	-	-	1.000
s38584-free	0	7338	5	0	-	0.999	-	-	0.999
wb_conmax-free	0	22182	0	0	-	1.000	-	-	1.000
EthernetMAC10GE-free	0	102966	1	0	-	1.000	-	-	1.000
Average	-	-	-	-	0.890	0.997	0.880	0.861	0.996

outperforms the method [17] in precision and the XGBoost + Trojan probability propagation method outperforms the method [17] in TPR.

Lastly, we compare the proposed method with the method [18]. In [18], 11 netlists were used for evaluation, all of which are included in the netlists in Table 3. As with the previous two methods, this method does not evaluate netlists containing more than 100K nets. As in the 2nd and 3rd rows of Table 9, even when our method takes into account all the 32 netlists, the XGBoost-only method is superior in terms of TNR and precision, and the XGBoost + Trojan probability propagation method is also superior in terms of TNR, precision, and F-measure, compared to the method [18].

The comparison results above are not completely fair since the used netlists are different from each other. However, the proposed method targets small to large-sized netlists including 100K or more nets, and still demonstrates the superiority in several aspects over the methods in [16]–[18].

6. Conclusion

In this paper, we first proposed the optimal set of HT features among many feature candidates at a netlist level for HT detection using XGBoost, one of the best gradient boosting decision tree models. Then we proposed an XGBoost-based HT detection method with optimized hyperparameters. The experimental results showed that the proposed method achieves the average F-measure of 0.842 for HT detection at Trust-HUB benchmark netlists. Furthermore, we applied the Trojan probability propagation method to the results of hyperparameter-tuned XGBoost. The results showed that the hyperparameter-tuned XGBoost with the Trojan probability propagation method achieves the average F-measure of 0.861, which is 0.194 points higher than that of the existing best method.

As in the netlists in Trust-HUB, HTs are located locally in HT-infected circuits. By using these Trojan properties, we will further improve the classification accuracy in the future.

Acknowledgments

These research results were obtained from the commissioned research (No. 05201) by National Institute of Information and Communications Technology (NICT), Japan.

References

[1] B. Liu and G. Qu, "VLSI supply chain security risks and mitigation techniques: A survey," Integration, vol.55, pp.438–448, 2016.

[2] K.G. Liakos, G.K. Georgakilas, S. Moustakidis, P. Karlsson, and F.C. Plessas, "Machine learning for hardware Trojan detection: A review," 2019 Panhellenic Conference on Electronics & Telecommunications (PACET), pp.1–6, 2019.

[3] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists based on machine learn-

ing," 2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS), pp.203–206, 2016.
[4] M. Oya, Y. Shi, N. Yamashita, T. Okamura, Y. Tsunoo, S. Goto, M. Yanagisawa, and N. Togawa, "A hardware-trojans identifying method based on Trojan net scoring at gate-level netlists," IEICE Trans. Fundamentals, vol.E98-A, no.12, pp.2537–2546, Dec. 2015.
[5] K. Hasegawa, M. Yanagisawa, and N. Togawa, "A hardware-Trojan classification method using machine learning at gate-level netlists based on Trojan features," IEICE Trans. Fundamentals, vol.E100-A, no.7, pp.1427–1438, July 2017.
[6] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks," 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS), pp.227–232, 2017.
[7] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–4, 2017.
[8] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.785–794, 2016.
[9] R. Negishi, T. Kurihara, and N. Togawa, "Hardware-Trojan detection at gate-level netlists using gradient boosting decision tree models," 2022 IEEE 12th International Conference on Consumer Electronics (ICCE-Berlin), pp.1–6, 2022.
[10] T. Kurihara and N. Togawa, "Hardware-Trojan classification based on the structure of trigger circuits utilizing random forests," 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp.1–4, 2021.
[11] "Htfinder," https://www.tjsys.co.jp/lsi/htfinder/index_j.htm
[12] "Trust-hub," https://trust-hub.org/
[13] C.H. Kok, C.Y. Ooi, M. Moghbel, N. Ismail, H.S. Choo, and M. Inoue, "Classification of Trojan nets based on SCOAP values using supervised learning," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–5, 2019.
[14] "XGBoost parameters — XGBoost 1.7.5 documentation," https://xgboost.readthedocs.io/en/stable/parameter.html
[15] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, vol.13, no.2, 2012.
[16] R. Sharma, N.K. Valivati, G. Sharma, and M. Pattanaik, "A new hardware Trojan detection technique using class weighted XGBoost classifier," 2020 24th International Symposium on VLSI Design and Test (VDAT), pp.1–6, IEEE, 2020.
[17] J. Chen, C. Dong, F. Zhang, and G. He, "A hardware-Trojans detection approach based on extreme gradient boosting," 2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET), pp.69–73, IEEE, 2019.
[18] Y. Zhang, S. Li, X. Chen, J. Yao, Z. Mao, J. Yang, and Y. Hua, "Hybrid multi-level hardware Trojan detection platform for gate-level netlists based on XGBoost," IET Computers & Digital Techniques, vol.16, no.2-3, pp.54–70, 2022.
[19] H. Chari, S. Aswale, V.N. Pawar, P. Shetgaonkar, and K. Chaman Kumar, "Advertisement click fraud detection using machine learning techniques," 2021 International Conference on Technological Advancements and Innovations (ICTAI), pp.109–114, 2021.
[20] K.R. Singh, R. Gupta, R.K. Kadian, and R. Singh, "An optimized XGBoost approach for predicting progression of hepatitis c using hyperparameter tuning and feature interaction constraint," 2022 2nd Asian Conference on Innovation in Technology (ASIANCON), pp.1–8, 2022.
[21] "Introduction to boosted trees — XGBoost 2.0.0-dev documentation," https://xgboost.readthedocs.io/en/latest/tutorials/model.html#learn-the-tree-structure

Appendix: XGBoost Algorithm and Feature Importance

The proposed HT detection method is based on XGBoost. In this appendix, we summarize the XGBoost algorithm and how it calculates the probability and feature importance briefly. See [8], [21] in detail.

XGBoost is one of the gradient boosting decision trees, a model that generates a number of decision trees and predicts a label [8], [21]. The level-wise tree growth is employed in XGBoost as a branch growth algorithm, where the branches grow in a breadth-first manner to a specified depth and the depth of all the leaf nodes becomes the same in the generated decision tree. Specifically, the branches grow as follows:

First, the objective function of XGBoost is the sum of the training loss and the regularization term, which can be approximated by

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^{n}\left[g_i\omega_{q(x_i)} + \frac{1}{2}h_i\omega^2_{q(x_i)}\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 \\ &= \sum_{j=1}^{T}\left[\left(\sum_{i\in I_j} g_i\right)\omega_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right)\omega_j^2\right] + \gamma T. \end{aligned} \tag{A·1}$$

In the above approximation, $\sum_{i=1}^{n}\left[g_i\omega_{q(x_i)} + \frac{1}{2}h_i\omega^2_{q(x_i)}\right]$ is the term of training loss and $\gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2$ is the regularization term. The variables used in Eq. (A·1) are summarized as follows:

- t shows the current iteration number when training the XGBoost model.
- n is the number of input data.
- T is the number of leaves in the XGBoost model.
- ω_j is the score of the leaf node j.
- $q(x_i)$ is a function assigning i-th data x_i to the corresponding leaf.
- I_j is the set of input data indices assigned to the j-th leaf.
- γ and λ are user-configurable parameters.
- $g_i = \partial_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$ and $h_i = \partial^2_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)}\right)$, where y_i and $\hat{y}_i^{(t-1)}$ are the objective value and the predicted value of the i-th data after $(t-1)$-th iteration, respectively, and l is the loss function.

Equation (1) can be further simplified by setting $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$.

$$Obj^{(t)} \simeq \sum_{j=1}^{T}\left[G_j\omega_j + \frac{1}{2}\left(H_j + \lambda\right)\omega_j^2\right] + \gamma T \tag{A·2}$$

From the above equation, the derivative of ω_j and the optimal objective function are obtained as:

$$\omega_j^* = -\frac{G_j}{H_j + \lambda} \tag{A·3}$$

$$Obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \tag{A·4}$$

The above equation measures how good a tree structure in the XGBoost model is. Minimizing the objective function optimizes the tree structure.

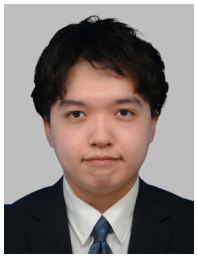
In the case of binary classification, the sum of the scores ω_j of the terminal leaves of each tree to which the test data belongs is calculated and then it is passed through a sigmoid function to obtain the *probability* that the test data belongs to one of the labels. In the XGBoost library, the *probability* can be calculated by the function *model.predict(test_data)* and it is mainly used in the proposed Trojan probability propagation method.

In XGBoost, an optimal tree is generated by optimizing one level of the tree at a time. When splitting a single leaf in the current XGBoost model into two leaves (left leaf and right leaf), the calculation of gain is as follows:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{A·5}$$

where G_L and H_L are G and H values in Eq. (A·2) of the new left leaf, respectively, and G_R and H_R are G and H values in Eq. (A·2) of the new right leaf, respectively. Gain is the score used in training to split a single leaf into two leaves. Among the possible new splits, the split with the maximum gain is chosen and the tree is extended.

Gain is also used for the calculation of feature importance. Each leaf has a corresponding feature, and the leaves retain the gain. In other words, the features have a corresponding gain for each leaf. *Feature importance* is given by the average gain across all the splits the feature is used in. In the XGBoost library, *feature importance* can be calculated by *model.get_score(importance_type="gain")* and this is used in Sect. 3.1.



Ryotaro Negishi received the B. Eng. degree from Waseda University in 2022 in computer science and communications engineering. He is presently working toward a master of engineering degree there. His research interests are hardware security, particularly hardware Trojans and the machine-learning-based detection.

Tatsuki Kurihara received the B. Eng. and M. Eng. degrees from Waseda University in 2020 and 2022 in computer science and engineering. His research interests are hardware security, particularly hardware Trojans and the machine-learning-based detection.

Nozomu Togawa received the B. Eng., M. Eng., and Dr. Eng. degrees from Waseda University in 1992, 1994, and 1997, respectively, all in electrical engineering. He is presently a Professor in the Department of Computer Science and Communications Engineering, Waseda University. His research interests include hardware security, quantum computation, and integrated system design. He is a member of IEEE and IPSJ.