Hardware Trojans Classification for Gate-level Netlists based on Machine Learning

Kento Hasegawa[†], Masaru Oya[†], Masao Yanagisawa[†], Nozomu Togawa[†]
[†]Department of Computer Science and Communications Engineering, Waseda University
Email: {kento.hasegawa, togawa}@togawa.cs.waseda.ac.jp

***Abstract*—Recently, we face a serious risk that malicious third-party vendors can very easily insert hardware Trojans into their IC products but it is very difficult to analyze huge and complex ICs. In this paper, we propose a hardware-Trojan classification method to identify hardware-Trojan infected nets (or Trojan nets) using a support vector machine (SVM). Firstly, we extract the *five hardware-Trojan features* in each net in a netlist. Secondly, since we cannot effectively give the simple and fixed threshold values to them to detect hardware Trojans, we represent them to be a *five-dimensional vector* and *learn* them by using SVM. Finally, we can successfully classify a set of all the nets in an unknown netlist into Trojan ones and normal ones based on the learned SVM classifier. We have applied our SVM-based hardware-Trojan classification method to Trust-HUB benchmarks and the results demonstrate that our method can much increase the true positive rate compared to the existing state-of-the-art results in most of the cases. In some cases, our method can achieve the true positive rate of 100%, which shows that all the Trojan nets in a netlist are completely detected by our method.**

***Index Terms*—hardware Trojan, gate-level netlist, machine learning, support vector machine (SVM), static detection**

I. Introduction

Due to globalization and cost-reduction in the IC market, IC designers often use third-party IC vendors. Some third-party IC vendors are not always reliable and they may embed or insert hardware Trojans into their IC products. Hardware Trojans can cause malfunctions, destroy the IC products, and/or leak secret information and how to detect them is a serious concern when designing and manufacturing ICs. In this paper, we focus on the hardware-Trojan detection at the design stage, particularly at gate-level netlists.

A. Existing Methods and their Problems

There is a vicious circle in hardware-Trojan detection [9]. Hardware-Trojan detection methods are roughly classified into the two types: the dynamic detection and the static detection.

a) Dynamic hardware-Trojan detection methods: The dynamic detection methods check whether a given circuit includes hardware Trojans or not, by simulating the circuit or actually running it. In these methods, it is necessary to find out the trigger states to activate hardware Trojans and see their behaviors. Since we require too much time to know the trigger states, we have to find out beforehand which part of the circuit includes hardware Trojans and set up test patterns very carefully [10]. Very recently, the pattern-matching-based dynamic method [4] and the clustering-based dynamic method [1] have been proposed.

b) Static hardware-Trojan detection methods: The static detection methods check whether a given circuit includes hardware Trojans or not, without simulating the circuit nor actually running it, but by just using hardware-Trojan related information. Since these methods do not actually run a circuit nor simulate it, we do not have to generate input test patterns and thus the detection results are not dependent on simulation results. Even if newly developed hardware

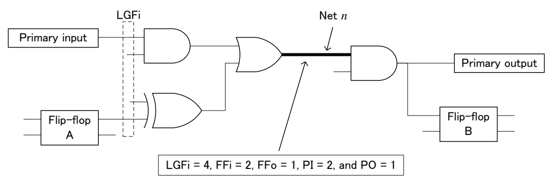


Fig. 1. The example of the Trojan feature values extracted from a netlist.

Trojans are given, the static detection methods can detect them by using existing hardware-Trojan related information. Recently, a static hardware-Trojan detection method [7] and a statistical technique for foundry identification [8] have been proposed. However, they are hard to apply to sequential circuits and large systems.

Overall, it is very difficult to develop a static hardware-Trojan detection approach at gate-level netlists but, if we can successfully develop this approach, it must be very effective to current and future hardware Trojans.

B. Summary of the Proposed Method

In the paper, we propose a static support-vector-machine-based (SVM-based) hardware-Trojan classification method at gate-level netlists. The proposed method classifies a set of the nets in a given netlist into Trojan nets and normal nets without using logic simulations nor functional simulations. First, we extract the five hardware-Trojan features, or *Trojan features*, based on the several known hardware-Trojan infected netlists. Then we apply *machine learning* to the extracted features. We consider the five Trojan features to be a *five-dimensional vector* and learn many five-dimensional vectors using a support vector machine (SVM). Finally, we can successfully classify a set of nets in a given unknown netlist into Trojan one and normal one by using the learned SVM classifier. Machine learning enables us to classify hardware Trojans automatically without simulating a given circuit nor actually running it.

C. Contributions of this Paper

The contributions of this paper are summarized as follows: 1) We find out the five hardware-Trojan features in netlists which contribute to detect hardware Trojans; 2) We propose an SVM-based hardware-Trojan classification method by learning the five Trojan features to classify a set of unknown nets into Trojan nets and normal nets. As far as we know, this is the world-first approach which successfully applies machine learning to detect hardware Trojans at gate-level netlist; 3) When applying our SVM-based hardware-Trojan classification method to Trust-HUB benchmarks, our method can much increase the true positive rate compared to the existing state-of-the-art results in most of the cases even though our method is completely static.

II. Hardware-Trojan Classification Method using SVM based on Trojan Features

We assume that hardware Trojans have several trigger conditions and they are activated when primary inputs and internal states of a given circuit satisfy the trigger conditions. Now we classify a set of

TABLE I
FIVE GATE-LEVEL NETLISTS FROM TRUST-HUB.

Data Name	Number of all the nets	Number of Trojan nets
RS232-T1300	307	9
RS232-T1500	314	12
s35932-T200	6435	16
s38584-T100	7399	9
s38584-T300	9110	1730

TABLE II
AVERAGE VALUES OF FFI, FFO, PI, AND PO.

Trojan/Normal	FFi	FFo	PI	PO
Trojan nets	1.11	1.51	4.60	3.65
Normal Nets	0.18	0.16	1.50	1.35

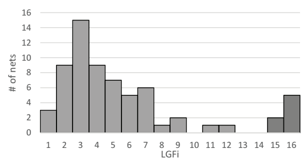


Fig. 2. The histogram of LGFi of Trojan nets.

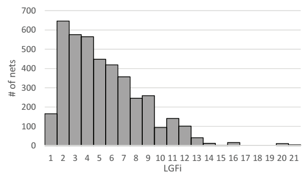
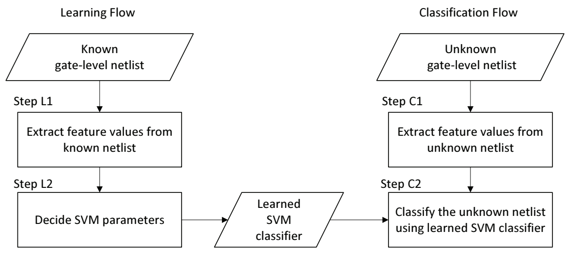


Fig. 3. The histogram of LGFi of normal nets.



Learning Flow
Known gate-level netlist
Step L1
Extract feature values from known netlist
Step L2
Decide SVM parameters
Learned SVM classifier
Classification Flow
Unknown gate-level netlist
Step C1
Extract feature values from unknown netlist
Step C2
Classify the unknown netlist using learned SVM classifier

Fig. 4. The flowchart of learning and classification.

nets in a given netlist into a set of Trojan nets and a set of normal nets. We focus on a static hardware Trojan detection approach not using circuit simulation. Note that, there are some reverse engineering methods targeting gate-level netlists such as in [6], but our method uses gate-level netlists themselves and do not use the other design information.

A. Trojan Feature Values in a Netlist

In this section, we first pick up hardware-Trojan infected gate-level netlists from Trust-HUB benchmark suites [11] and find out several net features which can be strongly related to hardware Trojans. Here we randomly pick up five hardware-Trojan infected netlists as listed in Table I.

Now we focus on a target net n in a given unknown netlist and extract several Trojan features of n.

1) Logic-gate fanins (LGFi): Since hardware Trojans are activated only when the primary inputs and/or internal states of the given circuit satisfy the trigger conditions, we expect that the transitive fanins of Trojan nets become large enough.

Figs. 2 and 3 summarize the breakdown of the fanin counts (logic-gate fanins, LFGi in short) in Trojan nets and normal nets in the five netlists of Table I, where LFGi here means the number of the inputs of the logic gates two-level away from the target net n (see LGFi in Fig. 1).As in Fig. 2, some of the Trojan nets have very large fanin counts compared to other Trojan nets. Those nets having large fanin counts are very likely to be Trojan nets and LFGi can be a large hint to classify between Trojan nets and normal nets.

However, some normal nets also have large fanin counts as in Fig. 3. The number of fanins itself is not enough to classify between Trojan nets and normal nets.

2) FFi, FFo, PI, and PO: Since some hardware Trojans have to memorize the internal states of a circuit, flip-flops are expected to be located near Trojan nets. Then we define FFi to be the minimum gate level to any flip-flop inputs from the target net n. We can also define FFo to be the minimum gate level from any flip-flop outputs to the target net n.

Some hardware Trojans give particular outputs from the chip and then the primary outputs are expected to be located near Trojan nets. Also, some hardware Trojans use primary inputs to activate hardware Trojans and then the primary inputs are expected to be located near Trojan nets. We define PI to be the minimum gate level from any primary input to the target net n. We can also define PO to be the minimum gate level to any primary output from the target net n.

In order to confirm the assumptions above, Table II summarizes the average values of FFi, FFo, PI, and PO in the five netlists of Table I. As in Table II, those values in Trojan nets are definitely smaller than those in normal nets. We expect that we can effectively classify between Trojan nets and normal nets by using FFi, FFo, PI, and PO values.

3) Trojan features: Overall, we can consider the five Trojan feature values below for every target net n in a netlist to classify between Trojan nets and normal nets:

1) LGFi (Logic Gate Fan-ins): The number of inputs of the logic gates two-level away from the net n.
2) FFi (FlipFlop Input): The number of logic levels to the nearest flip-flop input from the net n.
3) FFo (FlipFlop Output): The number of logic levels to the nearest flip-flop output from the net n.
4) PI (Primary Input): The minimum logic level from any primary input to the net n.
5) PO (Primary Output): The minimum logic level to any primary output from the net n.

Fig. 1 shows the example of the five feature values above extracted from a netlist. In this figure, we focus on the bold net n. Since n has the four transitive fanins as depicted in dotted lines in the figure, $\mathrm{LGFi} = 4$. Since the logic level from the flip-flop A to the net n is two, then $\mathrm{FFi} = 2$. Since the logic level from the net n to the flip-flop B is one, then $\mathrm{FFo} = 1$. In the same way, we have $\mathrm{PI} = 2$ and $\mathrm{PO} = 1$.

B. The Flow of the Proposed Method

In Section II-A, we extract the five feature values for every net n in a given netlist but we cannot set up simple and fixed threshold values for them to classify between Trojan nets and normal nets. Hence, we propose an SVM-based hardware-Trojan classification method, where it automatically learns Trojan nets and normal nets using the five Trojan feature values and classifies an unknown netlist into a set of Trojan nets and a set of normal nets by using the learned SVM classifier.

Fig. 4 shows the flow of the proposed method. The proposed method is composed of the two parts: the learning flow and the classification flow.

In the learning flow, we learn many known Trojan nets and normal nets by using the five Trojan feature values. The five Trojan feature values can be considered to be a *five-dimensional feature vector* $\boldsymbol{x}_n$ for every net n. Then we first extract the five-dimensional feature vector for every net in known netlists (Step L1). After that, we learn the extracted five-dimensional feature vectors using SVM (Step L2).

In the classification flow, we classify a given unknown netlist into a set of Trojan nets and a set of normal nets using the learned SVM classifier. We first extract the five-dimensional feature vector for each net from the unknown netlist (Step C1). After that, we identify each net in the unknown netlist to be a Trojan net or not by using the learned SVM classifier (Step C2).

In Step L2, We decide the parameter values γ and C in SVM [2] using learning data so that the true positive rate (TPR) is maximized where TPR is defined by $\mathrm{TPR} = \mathrm{TP}/(\mathrm{FN} + \mathrm{TP})$. In this equation,

TABLE III
THE TRUST-HUB DATA USED IN THE EXPERIMENTS.

Data Name	Number of normal nets	Number of Trojan nets	Data Name	Number of normal nets	Number of Trojan nets
RS232-T1000	266	45	s35932-T200	6419	16
RS232-T1100	300	12	s35932-T300	6423	37
RS232-T1200	305	10	s38417-T100	5807	12
RS232-T1300	298	9	s38417-T200	5807	15
RS232-T1400	299	12	s38417-T300	5807	44
RS232-T1500	302	12	s38584-T100	7390	9
RS232-T1600	298	9	s38584-T200	7380	200
s15850-T100	2429	27	s38584-T300	7380	1730
s35932-T100	6423	15			

TP shows the number of Trojan nets identified to be Trojan nets. FN shows the number of Trojan nets identified to be normal nets mistakenly. In Step L2, we firstly assume γ and C to be some values. Then we divide a set of all the learning data into the two sets; one set is known data and the other set is unknown data. By changing the γ and C values, we decide the best γ and C values which give the largest TPR value as the SVM parameters.

In Step C2, we classify the unknown netlist into a set of Trojan nets and a set of normal nets using the parameters γ and C.

III. HARDWARE-TROJAN CLASSFICATION RESULTS

In this section, we apply our proposed method to the several gate-level netlists in Trust-HUB benchmark suite [11]. We use an Intel Xeon E7-4870 computer environment. In our method, Step L1 and Step C1 are written in the C language and Step L2 and Step C2 are written in Python. We use Python machine learning library scikit-learn [5] for machine learning.

Table III summarizes the 17 gate-level netlists from Trust-HUB. Note that these data include very small number of Trojan nets and detecting them must be difficult.

A. Learning Flow

In (Step L1), we extract the five Trojan feature values for each net in a given netlist from Trust-HUB. In (Step L2), we decide the SVM parameters using the random search and gird search [3]. In the random search, we first give the rough parameter range R to C and γ and then execute machine leaning using the randomly selected values in R. As a result, we obtain the best C and γ values which maximize the TPR value. Now we pick up the parameter range R_{opt} surrounding the best C and γ values obtained by the random search. We perform more detailed search in this region by using the grid search.

B. Classification Flow

Once the parameter values are decided, we can perform the classification flow using the learned SVM classifier.

In (Step C1), we first extract the five Trojan feature values for every net in a given unknown netlist. In (Step C2), we classify the unknown netlist into a set of Trojan nets and a set of normal nets using the learned SVM classifier.

C. HTs Classification Results by Weighting Trojan Nets

We classify a netlist listed in Table III into a set of Trojan nets and a set of normal nets by using our proposed method. In the experiments, one of the netlists in Table III is considered to be an unknown netlist and the others are considered to be known netlists. After learning known netlists and setting up the learned SVM classifier, we classify an unknown netlist into a set of Trojan nets and a set of normal nets.

The number of Trojan nets is relatively small compared to the total number of nets in a given netlist. This is because malicious third-party vendors tend to hide their presence in IC and try to pass the IC tests. Hence learning data for Trojan nets in our SVM-based hardware-Trojan classification method may be much smaller than those for normal nets. It is very important to balance the learning data between Trojan nets and normal nets.

TABLE IV
LEARNED NORMAL NETS AND TROJAN NETS.

Unknown Data	No weighting		Static weighting		Dynamic weighting	
	Learned normal nets	Learned Trojan nets	Learned normal nets	Learned Trojan nets	Learned normal nets	Learned Trojan nets
RS232-T1000	63,067	2,169	63,067	43,380	7,225	7,175
RS232-T1100	63,033	2,202	63,033	44,040	7,191	7,038
RS232-T1200	63,028	2,204	63,028	44,080	7,204	7,153
RS232-T1300	63,035	2,205	63,035	44,100	7,202	7,130
RS232-T1400	63,034	2,202	63,034	44,040	7,218	7,130
RS232-T1500	63,031	2,202	63,031	44,040	7,215	7,176
RS232-T1600	63,035	2,205	63,035	44,100	7,205	7,130
s15850-T100	60,904	2,187	60,904	43,740	6,664	6,490
s35932-T100	56,910	2,199	56,910	43,980	7,155	6,969
s35932-T200	56,914	2,198	56,914	43,960	7,219	6,923
s35932-T300	56,910	2,177	56,910	43,540	7,215	6,975
s38417-T100	57,526	2,202	57,526	44,040	7,209	6,992
s38417-T200	57,526	2,199	57,526	43,980	7,213	6,946
s38417-T300	57,526	2,170	57,526	43,400	7,204	7,072
s38584-T100	55,943	2,205	55,943	44,100	6,663	6,426
s38584-T200	55,953	2,014	55,953	40,280	7,225	7,200
s38584-T300	55,953	484	55,953	9,680	7,225	7,068

Based on this discussion, we have performed the three types of experiments as follows:

1) **No weighting:**
We have just used original data for learning. For example, if some known netlist has N_n normal nets and N_t Trojan nets, SVM has learned N_n normal nets and N_t Trojan nets.
2) **Static weighting** ($W = 20$)**:**
Every Trojan net is learned by SVM W times. For example, if some known netlist has N_n normal nets and N_t Trojan nets, SVM has learned N_n normal nets and $W \times N_t$ Trojan nets. In this experiment, we set $W = 20$.
3) **Dynamic weighting:**
We balance the number of learned normal nets and the number of learned Trojan nets. At first, we find out the nets which have the identical feature vector in normal nets, leave one of them and delete the rest of them. We also find out the nets which have the identical feature vector in Trojan nets, leave one of them and delete the rest of them. For example, if the feature vector $(1, 2, 1, 2, 2)$ appears three times in normal nets, we delete two of them and leave one of them.
After that, we balance the number of learned normal nets and the number of learned Trojan nets as follows: Assume that we now have N'_n normal nets and N'_t Trojan nets. Then SVM has learned every normal nets once but every Trojan net (N'_n/N'_t) times. Totally, SVM has learned N'_n normal nets and N'_n Trojan nets.

Table IV summarizes the number of learned normal nets ("learned normal nets") and the number of learned Trojan nets ("learned Trojan nets") in each experiment type when classifying every unknown data. For example, when RS232-T1000 is considered to be an unknown netlist, all the other 16 netlists are considered to be known netlists. Then, in case of "No weighting", the SVM classifier learns 63,067 normal nets and 2,169 Trojan nets from the 16 netlists. Since "Dynamic weighting" deletes Trojan feature vectors identical to both normal nets and Trojan nets, the number of learned normal nets and the number of learned Trojan nets are reduced compared to the other two experiments but, as shown below, the accuracy of the SVM classifier in this case is much increased by balancing normal nets and Trojan nets and deleting ambiguous Trojan feature vectors.

Table V shows the classification results. In this table, TP shows that the number of normal nets identified to be normal nets. FN shows the number of Trojan nets identified to be normal nets mistakenly. TPR shows the true positive rate. TNR shows the true negative rate which is defined by the the number of the true negatives over the number of total normal nets, where the true negatives here mean the normal nets which are identified to be the normal nets.

As in the table, if we just use original data by giving no weights ("No weights"), TPR becomes 0 which shows that all the Trojan nets are identified to be normal nets mistakenly. By giving a static weight ($W = 20$) to every Trojan net ("Static weight"), TPR values are improved. By giving a dynamic weight and balancing

TABLE V
EXPERIMENTAL RESULTS.

Unknown	No weighting						Static weighting						Dynamic weighting					
Data	C	γ	FN	TP	TPR	TNR	C	γ	FN	TP	TPR	TNR	C	γ	FN	TP	TPR	TNR
RS232-T1000	1	0.001	45	0	0%	100%	1	0.001	41	4	9%	57%	1	0.001	21	24	53%	31%
RS232-T1100	1	0.001	12	0	0%	100%	1	0.001	9	3	25%	63%	1	0.001	5	7	58%	27%
RS232-T1200	1	0.001	10	0	0%	100%	1	0.001	8	2	20%	63%	1	0.001	2	8	80%	26%
RS232-T1300	1	0.001	9	0	0%	100%	1	0.001	8	1	11%	61%	1	0.001	1	8	89%	26%
RS232-T1400	1	0.001	12	0	0%	100%	1	0.001	10	2	17%	59%	1	0.001	2	10	83%	22%
RS232-T1500	1	0.001	12	0	0%	100%	1	0.001	10	2	17%	61%	1	0.001	2	10	83%	24%
RS232-T1600	1	0.001	9	0	0%	100%	1	0.001	9	0	0%	61%	1	0.001	1	8	89%	26%
s15850-T100	1	0.001	27	0	0%	100%	1	0.001	24	3	11%	89%	1	0.001	2	25	93%	66%
s35932-T100	1	0.001	15	0	0%	100%	1	0.001	11	4	27%	92%	1	0.001	1	14	93%	60%
s35932-T200	1	0.001	16	0	0%	100%	1	0.001	13	3	19%	91%	1	0.001	0	16	100%	59%
s35932-T300	1	0.001	37	0	0%	100%	1	0.001	17	20	54%	91%	16	0.001	27	10	27%	58%
s38417-T100	1	0.001	12	0	0%	100%	1	0.001	11	1	8%	92%	1	0.001	0	12	100%	76%
s38417-T200	1	0.001	15	0	0%	100%	1	0.001	15	0	0%	92%	1	0.001	4	11	73%	76%
s38417-T300	1	0.001	44	0	0%	100%	1	0.001	0	44	100%	92%	11	0.001	0	44	100%	72%
s38584-T100	1	0.001	9	0	0%	100%	1	0.001	4	5	56%	91%	1	0.001	0	9	100%	62%
s38584-T200	1	0.001	200	0	0%	100%	1	0.001	44	156	78%	92%	1	0.001	12	188	94%	64%
s38584-T300	1	0.001	1,730	0	0%	100%	1	0.001	618	1,112	64%	98%	1	0.001	185	1,545	89%	66%

TABLE VI
COMPARISON BETWEEN [1] AND THE OUR PROPOSED METHOD (WITH DYNAMIC WEIGHTING).

Data	TPR in [1]	TPR in ours	TNR in [1]	TNR in ours
s15850-T100	61%	93%	99%	66%
s35392-T200	27%	100%	99%	59%
s38417-T100	100%	100%	99%	76%
s38584-T200	99%	94%	98%	64%

the learned normal nets and Trojan nets ("Dynamic weight"), we can have the best TPR values. By using our method with dynamic weighting, we can achieve 80% or more TPR values in most of the cases. In some cases, we can achieve 100% TPR values. The results clearly demonstrate that our method with dynamic weighting can successfully find out almost all the Trojan nets in a given unknown netlist by just using learning data. TNR in dynamic weighting is relatively small compared to static weighting. However, we believe that identifying Trojan nets to be Trojan nets is the most important in Trojan detection, since we can investigate all the Trojan nets and gates around the identified Trojan nets.

In all the cases, Step L1 and Step L2 require three to ten hours and Step C1 and Step C2 require one to two hours.

D. Comparison to the Existing Method

Table VI shows the comparison results between the method proposed in [1] and our proposed method with dynamic weighting. The results of [1] are just cited from [1]. The method in [1] uses signal correlation between Trojan nets and normal circuits and identify whether each net in an unknown netlist is Trojan or not, which is one of the most strong hardware Trojan detection methods proposed so far. Table VI shows that our method with dynamic weighting outperforms the method in [1] in most of the cases in terms of the TPR.

Since the method in [1] is based on functional simulation, TPR value in this method can be much dependent on the functional simulation as well as input patterns. Particularly in large netlists, it is almost impossible to run functional simulation by giving all the possible input patterns.

On the other hand, our proposed method does not require functional simulation nor logic simulation. The results obtained by our method is very static and thus, even if a large circuit is given, our method can identify a Trojan net just based on the learned SVM classifier.

IV. CONCLUSIONS

In the paper, we have proposed an SVM-based hardware-Trojan classification method for gate-level netlists by machine learning. We have first extracted the five Trojan feature values from get-level netlists and optimized the SVM parameters using learning data. Then, we have classified unknown netlists and identified hardware Trojans using the learned SVM classifier.

The experimental results demonstrate that the true positive rate of the proposed method is increased to up to 100%. Even if the proposed method is a completely static method which does not use any logic/functional simulations, the results are almost equal or better than those obtained by the existing state-of-the-art dynamic detection method in terms of TPR.

In the future, we will improve the proposed method so as to increase the true positive rate and decrease the false positive rate so that we can completely extract all the hardware-Trojan parts from a given netlist.

ACKNOWLEDGMENTS

This study was supported in part by the MIC/SCOPE #141303001.

REFERENCES

[1] B. Cakir and S. Malik, "Hardware trojan detection for gate-level ics using signal correlation based clustering," in *Proc. Design, Automation and Test in Europe (DATE)*, pp. 471–476, 2015.
[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
[3] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification." https://www.cs.sfu.ca/people/Faculty/teaching/726/spring11/svmguide.pdf, 2003.
[4] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, "A score-based classification method for identifying hardware-trojans at gate-level netlists," in *Proc. Design, Automation and Test in Europe (DATE)*, pp. 465–470, 2015.
[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011.
[6] E. Tashjian and A. Davoodi, "On using control signals for word-level identification in a gate-level netlist," in *Proc. Design Automation Conference (DAC)*, pp. 1–6, 2015.
[7] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (ACM-CCS)*, pp. 697–708, 2013.
[8] J. B. Wendt, F. Koushanfar, and M. Potkonjak, "Techniques for foundry identification," in *Proc. Design Automation Conference (DAC)*, pp. 1–6, 2014.
[9] J. Zhang, F. Yuan, and Q. Xu, "Detrust: defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (ACM-CCS)*, pp. 153–166, 2014.
[10] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 67–70, 2011.
[11] "Trust-hub." http://www.trust-hub.org.