Hardware Trojan Detection Using XGBoost Algorithm for IoT with Data Augmentation Using CTGAN and SMOTE

C. G. Prahalad Srinivas(✉), S. Balachander, Yogesh Chandra Singh Samant, B. Varshin Hariharan, and M. Nirmala Devi

Department of Electronics and Communication Engineering, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, Coimbatore, India
cb.en.u4ece17239@cb.students.amrita.edu, m_nirmala@cb.amrita.edu

Abstract. Internet of things is the next blooming field in engineering. Seamless connectivity has become the new normal when devices are manufactured. This demand for IoT (Internet of Things) also results in various threats. Hardware trojans are one such threat which is encountered. Hardware trojan, being sneak circuits are overlooked when a new chip is manufactured. This further solidifies the threat posed by the hardware trojans. Applying machine learning model directly to the dataset will prove futile, since the amount of trojan infected nets are very minimal compared to the regular nets. In this work, this problem is addressed by creating a machine learning model, which can handle the data imbalance, using CTGAN (Conditional Tabular Generative Adversarial networks) and SMOTE (Synthetic Minority Oversampling Technique) algorithms, and detect the trojan infected nets.

Keywords: Hardware Trojan · SMOTE · Machine learning · CTGAN · XGBoost

1 Introduction

Hardware trojans are sneak circuits which can be of different types and can perform different functions. These trojans are also occupy minimal area and hence are incredibly hard to detect manually. The insertion of these trojan can happen at several stages during fabrication. This increases the difficulty in detecting the presence of a trojan in a chip.

In this work, an approach using machine learning is suggested to solve this problem. Despite having neural network and deep learning techniques, machine learning algorithms were chosen for the detection because of its speed of training and execution. Since this work deals with IoT (Internet of Things), the detection needs to happen almost instantaneously, and training also needs to be done very quickly if a new circuit is being introduced. This is the main reason for choosing machine learning algorithms over neural networks.

The model that has been chosen for hardware trojan detection is XGBoost (eXtreme Gradient Boosting). It is currently considered one of the best machine learning models.

N. Kumar et al. (Eds.): UBICNET 2021, LNICST 383, pp. 116–127, 2021.
https://doi.org/10.1007/978-3-030-79276-3_10

It has also been proven to be very effective for the problem at hand [1, 2]. XGBoost was designed for its swiftness over the normal gradient boosting algorithm and is able to consistently outperform other models.

Another big problem faced when it comes to trojan detection is the lack of significant trojan nets. Since the number of nets are too small, the model gives more importance to the non-trojan nets and hence is not able to detect the trojan nets accurately. This is why two of the widely used synthetic data generation techniques are being tested and evaluated, namely CTGAN (Conditional Tabular Generative Adversarial networks) and SMOTE (Synthetic Minority Oversampling Technique).

Finally, the results of CTGAN and SMOTE are being compared. For the purpose of this comparison, the main metric is taken to be recall and not accuracy. The reason for this being that recall is able to convey how many relevant data points are available in the dataset. Since the data is being synthetically generated, recall will provide a good measure for comparison.

This work starts by having a look at previous solutions for the problem. A basic understanding of hardware trojans and the types of hardware trojans used for this work is then provided. Next, the methodology of the experiments conducted is discussed. It includes the feature extraction process, the data augmentation process and the machine learning model building process. Finally, the results obtained by the experiment is analyzed and the final conclusion is drawn.

2 Related Works

Similar Approaches for Different Problems

In the [1] by Richa Sharma et al., they have suggested a Hardware trojan detection technique using Weighted XGBoost Classifier. The authors have extracted observability and controllability measures for all the nets and have suggested that the trojan nets display a higher observability and controllability measure. They have also suggested a permutation-based feature selection for this problem, which was found to be interesting. Cheng Dong et al. [2] have taken an approach of hardware trojan detection towards the IoT field. They have suggested that the IoT is the next big thing and hardware chips and ICs will be a very integral part of this process. They have used XGBoost for hardware trojan detection and have achieved an accuracy of 99.83%. The paper [3] provides an overview on the XGBoost algorithm and the mathematics behind XGBoost.

In this work, a different set of features has been considered to try and improve the accuracy. Synthetic data generation will also be used to try and improve the dataset and hence improve the accuracy of the model. This work was tested with 7 different circuits with 3 different trojans inserted to ensure that this approach produces consistent results for a wide range of circuits.

Different Approaches for Similar Problems

In [4], the author detects trojans using a compressive sensing algorithm to detect hardware trojans. The author talks about generating test patterns which can effectively trigger the trojans into discharging their payload. This method can be considered as a very good

replacement for the golden chip method of trojan detection. In [5] the author is suggesting a way to detect trojans using weighted average voting. Here, the authors send the same set of inputs to different modules of the ISCAS (IEEE International Symposium on Circuits and Systems) circuits and then considers the history of these outputs to detect if a trojan is present or not. [6] Discusses about detecting trojans using bitstreams of output data obtained by using different modules of the same circuit. If there were any viruses, then the bitstream is going to become abnormal, hence showing that the particular circuit has an infection. [7] uses leakage power calculation for multiple iterations of the circuit to observe potential changes and subsequently detect the hardware trojan inserted circuits. Even though the above methods can successfully detect trojans, they will not be able to point to the exact net that the trojan is present.

3 Hardware Trojans and Its Types

Hardware trojans are small bits of circuit that can decrease the performance of a given circuit. These trojans are sometimes also used to steal valuable information or data.

3.1 Threat Model

Trojans need two specific ingredients for it to function properly. One of them is a trigger and another is a payload. A trigger in a trojan is a circuit which will turn on the trojan at the right point of time. It is important for hardware trojans to not be detected during testing phase. That is why the trigger circuit will wait for the right condition to trigger the trojan circuit. This right condition will depend on the designer who is designing the trojan. Some triggers will wait for a right set of inputs, while some may trigger after certain amount of time.

The payload of a trojan is the part of the trojan circuit which actually causes harm to the host circuit. These can vary from changing the functionality of the circuit to extending the time taken to produce the desired output. The trojans can be vastly classified based on the types of triggers and the types of payload [8]. Gate level netlist is considered as an input, and 3 types of trojans are analyzed.

3.2 Trojans in IoT

The significance of hardware trojans on IoT is that most of IoT systems are realized using hardware deployment. And where there is hardware, there is always the risk of having a hardware trojan. Starting from a small LoRa (Long Range) module to a node MCU (Micro Controller Unit), a hardware trojan can be present everywhere. Since these products are bought by the customers and the distributors have a hard time checking and removing the trojans, the trojans inserted by a manufacturer may go unnoticed. That is why, in this work, Trojan detection will be made possible with a cheap yet effective machine learning based method.

3.3 The Types of Trojans Considered for Testing

Three types of hardware trojans are considered for detection. Namely: Always on, sequential and combinational trojan.

Always on Trojans
An always on trojan is, as the name suggests, always on. These are the only type of trojans that don't require a trigger circuit to turn them on. Thus, identifying the trojans is a bit more difficult because the amount of circuitry involved can be as small as a single gate.

Sequential Trojans
Sequential trojans [9] are the trojans which are activated using a sequential circuit. This circuit is commonly an asynchronous up counter which will trigger the circuit when it reaches a certain value, or after a certain number of a single operation is performed. Another example of a sequential trojan is something known as a "time bomb". These trojans keep track of time and once the time is reached, they release the payload into the host.

Combinational Trojans
Combinational trojans are trojans which are activated using a combinational circuit. Common examples of combinational trojans are FSM (Finite State Machine) trojans. These trojans have an FSM built as a trigger and will trigger the payload only if a certain pattern of inputs is received. The probability of receiving that specific pattern during the testing phase is very small and hence these trojans are effective at avoiding detection during testing phase.

4 Methodology

4.1 Feature Extraction

Several features have been extracted from the gate level netlists. Seven different circuits with 3 different trojans are considered. Table 1 contains the details of the benchmark circuits which were considered.

The features which have been extracted from these circuits are described in Table 2.

The dataset with which this experiment is done, in general has two types of features the first one is discrete and the other is continuous. In the datasets which are considered, the following has been observed:

Features like Level, Connectivity, Po, Score, Fan-in, Load, Resistance, Pins, Total net are considered as discrete features, since the level of the net from the input layer in any given circuit comes from fixed set of values. For connectivity, the number of gates the given net is connected, comes from fixed set. For Po, the number of levels the net is away from the output net, comes from the fixe set similar to Level. Score is summation of connectivity and Po, hence it is also considered as discrete feature. The same applies to Fan-in, and Fan-out. Generally, a net consists of combination of gates, hence each gate has its resistance and load resistance. Hence Resistance and Load resistance are

Table 1. List of benchmark circuits and the type of inserted trojan

Circuit name	Always on trojan	Sequential trojan	Combinational trojan
C17	Yes	Yes	Yes
C432	Yes	Yes	Yes
C1908	No	Yes	Yes
C6288	Yes	Yes	Yes
S27	Yes	Yes	Yes
S298	Yes	Yes	Yes
S820	Yes	Yes	Yes

Table 2. List of features extracted

Feature Name	Description
Level	The level of the net from the input layer
Connectivity	The number of gates the given net is connected to
Po	The number of levels the net is away from the output nets
Score	The sum of level, connectivity and po
Fan-in	The number of incoming nets into the gate
Load	Capacitive load of the net
Resistance	Resistive load
Pins	Number of pins
Total net	Maximum nets available
Static Prob	Probability that a certain net will stay in the current state
Toggle rate	Probability that a certain net will switch from current state
Switching power	The power needed to switch the net from one state to another

considered as discrete features. And Total resistance is sum of Load resistance and circuit resistance, again it is also considered as discrete feature.

But features like Static probability, Toggle rate and switching power are considered as continuous. Since static probability means the time duration for which the net will be at logic '1'. It varies from net to net and hence it doesn't come from a set of values. Toggle rates means the number of times the signal will change its state, hence it doesn't take any fixed set of values. Switching power too doesn't come from a fixed set of values. Hence the discussed 3 features can be collectively grouped as Continuous features.

4.2 Handling Data Imbalance Problem

A distinct problem was encountered when extracting the features. That problem is that the amount of nets which do not have trojan far outweigh the nets which are trojan infected. This creates a data imbalance problem, that is, the amount of data for one particular class is greater than that of the other class.

To mitigate this problem, two algorithms are being used: SMOTE Algorithm and CTGAN algorithm. These two algorithms are used to augment the dataset so that the amount of trojan infected and trojan free nets become the same.

CTGAN

Conditional Tabular Generative Adversarial Network is method of modelling tabular data by involving mode specific normalization, which addresses Non- Gaussian and multimodal distribution of continuous columns and conditional generator and training by sample, which address the imbalanced discrete columns [10].

Mode Specific Normalization

Each continuous column is modelled using Variational Gaussian Mixture Model to estimate the number of modes and fit a mixture model. For each value in a continuous column, the probability of that value coming from each mode is calculated. Sample one mode from the given probability density and used that mode to normalize the value. Then a row could be expressed as a concatenation of continuous and discrete columns.

$$N_{i,j} = \frac{(c_{i,j} - \eta_n)}{4\,\Phi_n} \tag{1}$$

Where,

$N_{i,j}$ is the Normalized value at i^{th} column and j^{th} row.
$c_{i,j}$ is the value at i^{th} column and j^{th} row.
η_n is the n^{th} Gaussian mode in i^{th} column.
Φ_n is the standard deviation of n^{th} Gaussian distribution in i^{th} column.

Conditional Generator and Training by Sample

In a traditional GAN (Generative Adversarial Network) approach, the vectored input is fed to the discriminator without taking into consideration for the imbalance in the discrete columns (categorical columns). While sampling the data when rows of one particular category isn't sufficiently represented, the generator won't give the desired output, i.e. the generator is not trained correctly. The generator can be interpreted as the conditional distribution of rows given that particular value at that particular column,

$$s \sim P(row|D_i = v), \tag{2}$$

The original data distribution can be reconstructed as follows

$$P(row) = \sum_{val \in D_i} P(row|D_i = v)P(D_i = val) \tag{3}$$

Where,

s is the generated sample,
D_i is the i^{th} discrete column and.
v is value at the i^{th} discrete column.

The output produced by the generator is to be assessed by the discriminator. The sampling of real training data and construction of the condition vector (concatenation of one hot encoded of discrete columns based on the requirements) should comply so that the discriminator could estimate the distance. A good sampled condition vector and training data could almost explore all the values in discrete columns.

SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is a technique used to address the data imbalance problem by oversampling the minority class. This data augmentation method works on the principle of nearest neighbours, where new data points will be created by combining two or more original instances. When a random datapoint *a* is being considered from the minority class on the feature space, the *k* of the nearest neighbours is found for that particular datapoint. A random nearest neighbour *b* is selected, and an imaginary line *l* is drawn in-between the two points *a* and *b*. New instances are added or sampled on that line *l*, as these new synthetic datapoints are a combination of *a* and *b*. This process is repeated for *k* different datapoints that are near to *a*.

By expanding this process to the other instances in the minority class, multiple duplicates can be created, thus solving the data imbalance setback. This algorithm is effective because of the fact that the newly created instances will be closer the existing instances in the feature space. This helps the classifier to build a larger decision region that contains the datapoints belonging to the minority class.

For the dataset that is being considered for this application, 15 different features are present in it. The above concept will be expanded to a 15-dimensional vector space, where a specific *a* and *b* will correspond to a single datapoint, and will be placed in a unique position. However, the type of each feature can be either continuous or discrete as mentioned in the feature extraction section. When the oversampling takes place, the continuous feature values are taken as such, since they carry the freedom to have a value chosen in-between two samples. This isn't the case with the discrete values, since it is rigid as the feature carries specific values. To overcome this, SMOTE considers the discrete values as continuous while oversampling, but rounds off the oversampled values to the nearest integer when it comes to discrete features.

The realization of this method is achieved in python with the help of *imblearn* library, which contains the model for SMOTE under the sub-library *over_sampling*. The dataset with all the features that are both continuous and discrete was pushed into the function as a whole. The function has the capability to identify the minority class and the number of samples required to balance the dataset. Also, the neighbour value *k* can be tuned accordingly on top of the default value to produce the required number of samples. This advantage was used to create appropriate samples in the dataset containing limited samples, as the value *k* was minimized.

Given below are the results obtained after applying SMOTE algorithm to the considered dataset (Table 3).

Table 3. Comparison of instances before and after applying SMOTE

Circuit	Trojan type	Instances before SMOTE			Instances after SMOTE		
		0	1	Total	0	1	Total
C17	Always on	7	9	16	9	9	18
	Combinational	5	8	13	8	8	16
	Sequential	6	11	17	11	11	22
C432	Always on	116	85	201	116	116	232
	Combinational	86	113	199	113	113	226
	Sequential	107	88	195	107	107	214
C1908	Combinational	632	283	915	632	632	1264
	Sequential	706	208	914	706	706	1412
C6288	Always on	2133	320	2453	2133	2133	4266
	Combinational	2035	413	2448	2035	2035	4070
	Sequential	2251	198	2449	2251	2251	2502
S27	Always on	11	10	21	11	11	22
	Combinational	7	11	18	11	11	22
	Sequential	10	12	22	12	12	24
S298	Always on	92	49	141	92	92	184
	Combinational	86	53	139	86	86	172
	Sequential	100	41	141	100	100	200
S820	Always on	253	55	308	253	253	506
	Combinational	246	63	309	NA*		
	Sequential	264	49	313	264	264	528

*Note that S280 did not produce synthetic data due to the unavailability of nearest neighbours.

XGBoost Algorithm

eXtreme Gradient Boosting algorithm has been used for training the model over the dataset. This algorithm comes under the umbrella of ensemble models. This machine-learning model are algorithm which uses a weaker classifier to make itself stronger. There are broadly two types of ensemble algorithms, bagging and boosting. The XGBoost algorithm comes under the boosting category. It is a decision tree-based ensemble machine learning technique that uses the Gradient boosting framework for machine learning prediction.

XGBoost tries to find the optimal output value for a tree f_t in an iteration t that is added to minimize the loss function across all data point (4) [3].

$$L(t) = \sum_{i=1}^{n} l\left(y_i,\ \hat{y}^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) \tag{4}$$

Where,

l represents a differentiable convex loss function
$\hat{y}_i$ is the prediction
y_i is the target

The steps to perform XGBoost are as follows:

1. For Initial predictions and the corresponding initial residual errors, the mean of target values is calculated.

$$h_0(x) = \text{Mean}(Y) \tag{5}$$

$$\hat{Y} = Y - h_0(x) \tag{6}$$

Where,

h_0 represents the initial prediction
$\hat{Y}$ represents the initial residual errors
Y represents the target values

2. A model is trained with independent variables and residual errors which is used to predict the output.
3. The output obtained from the model is used for additive predictions and the calculation of residual errors with some learning rate as shown in the following images.

$$\text{Gain} = \text{Left}_{\text{similarity}} + \text{Right}_{\text{similiarity}} - \text{Root}_{\text{similarity}} \tag{7}$$

$$\textit{Similarity Score} = \frac{\left(\sum \textit{Residual}_i\right)^2}{\sum \left[\textit{previous probablity}_i \text{ x } (1 - \textit{previous probablity}_i)\right] + \lambda} \tag{8}$$

$$\textit{Output} = \frac{\left(\sum \textit{Residual}_i\right)}{\sum \left[\textit{previous probablity}_i \text{ x } (1 - \textit{previous probablity}_i)\right] + \lambda} \tag{9}$$

Where,

λ represents the regularization factor.

4. Steps 2 and 3 are repeated for M times until the specified number of models are built.
5. The final prediction from boosting is that adding the sum of all previous predictions made by the models.

$$F_{final} = h_0(x) + \alpha \sum h_i(x) \tag{10}$$

Where,

F_{final} is the final prediction
$h_0(x)$ is the initial prediction
$h_i(x)$ is the previous predictions
α is the learning rate

XGBoost is a very reliable, powerful, and sophisticated algorithm that offers the state-of-the-art results for several complex problems.

5 Results

Table 4. Results after applying CTGAN and SMOTE algorithms on the dataset

Circuit	Trojan type	Recall before augmentation (%)		Recall after SMOTE (%)		Recall after GAN (%)		Accuracy of XGBoost classifier (%)		
		0	1	0	1	0	1	Before data synthesis	After SMOTE	After CTGAN
C17	Always on	67	100	80	100	86	100	83	83	88
	Combinational	0	75	67	67	94	100	60	67	94
	Sequential	0	80	100	100	100	75	67	100	97.5
C432	Always on	94	79	94	89	93	98	87	91	97.7
	Combinational	68	58	65	71	99	66	62	68	95
	Sequential	98	96	90	97	98	100	97	93	100
C1908	Combinational	99	95	98	98	100	96	97	98	100
	Sequential	98	97	98	97	98	100	98	98	100
C6288	Always on	97	57	93	94	99	99	92	93	99
	Combinational	95	58	94	86	88	100	90	90	98
	Sequential	100	17	88	93	100	100	94	91	100
S27	Always on	50	100	60	100	100	100	71	75	100
	Combinational	100	75	80	100	97	75	83	88	95
	Sequential	75	100	100	40	100	50	88	62	98
S298	Always on	94	67	83	77	89	99	85	80	98
	Combinational	75	83	75	67	100	97	78	70	97
	Sequential	88	60	94	88	92	99	79	91	99
S820	Always on	91	60	91	94	99	100	81	92	100
	Combinational	95	28	NA		97	99	83	NA	98
	Sequential	96	54	87	82	96	100	90	85	99

From the above results, it can be concluded that the CTGAN is better than the SMOTE algorithm for the problem on hand (Table 4). It can also be noticed that the SMOTE algorithm was unable to generate synthetic data for S820 combinational trojan circuit. This was because there was not sufficient minority neighbours for SMOTE algorithm to work on. The CTGAN algorithm on the other hand is robust and is able to overcome the shortcoming of the SMOTE algorithm.

This is one of the first works which use recall as a metric for augmentation. Recall explains about how many relevant items are being selected in the dataset. True positive explains when a sample from class A is predicted to be in class A, and when the same sample is predicted to be in class B it is false negative. Obtaining a recall of 100% reveals

that there are no false negatives, and all the samples that are being selected are relevant to its particular class.

Since, after all, augmentation technique introduce new datapoints, it is vital for the datapoints to be relevant to the particular class. This is the reason why recall was chosen for evaluation. The higher the recall value, the more relevant the generated datapoints are.

The average recall has been calculated and tabulated in Table 5.

Table 5. Comparison of average recall values

	Before augmentation	SMOTE	CTGAN
Average Recall 0	79	86.157	96.25
Average Recall 1	71.95	86.315	92.65

From Table 5, it can be clearly observed that the average recall of CTGAN is far better than the average recall of SMOTE. Although SMOTE has made a huge jump in average recall when comparing with the recall before augmentation, CTGAN has achieved an even higher recall than SMOTE.

Similarly, the average accuracy before augmentation was 83.25% and after using SMOTE was 85% and with CTGAN, it was at its highest at 97.66%, with 6 circuits achieving a perfect 100% accuracy.

6 Conclusion

From the above results, it can be concluded that resolving the data imbalance problem is improving the accuracy and the recall of the model. These trojan detection techniques can be applied to any chips and when CTGAN approach is used, accuracy of almost 100% can be achieved. Also once trained and deployed, this model can be used over and over for testing IoT chips. This helps the distributors to detect trojans with very less resources, time and money and help the customers by providing them trojan free ICs (Integrated Circuits) for use in IoT.

In future this approach can be experimented for trust benchmark circuits and check if they are also able to produce good results. There is also a scope for fusing two or more models to get 100% accuracy for all the circuits.

References

1. Richa, S., et al.: A new hardware trojan detection technique using class weighted XGBoost classifier. In: 2020 24th International Symposium on VLSI Design and Test (VDAT). IEEE (2020)
2. Chen, D., et al.: A machine-learning-based hardware-Trojan detection approach for chips in the Internet of Things. Int. J. Distrib. Sensor Netw. **15**(12), 1550147719888098 (2019)

3. Tianqi, C., Guestrin, C.: Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining (2016)
4. Priyatharishini, M., Nirmala Devi, M.: A compressive sensing algorithm for hardware trojan detection. Int. J. Electr. Comput. Eng. **9**(5), 4035 (2019)
5. Aishwarya, G., et al.: Virtual instrumentation-based malicious circuit detection using weighted average voting. In: Microelectronics, Electromagnetics and Telecommunications, pp. 423–431. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7329-8_43
6. Mohankumar, N., Jayakumar, M., Nirmala Devi, M.: CRC-based hardware trojan detection for improved hardware security. In: Microelectronics, Electromagnetics and Telecommunications, pp. 381–389. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7329-8_39
7. Priya, S.R., et al.: Hardware malicious circuit identification using self-referencing approach. In: 2017 International Conference on Microelectronic Devices, Circuits and Systems (ICMDCS). IEEE (2017)
8. Ramesh, K., et al.: Trustworthy hardware: identifying and classifying hardware trojans. Computer **43**(10), 39–46 (2010)
9. Xinmu, W., et al.: Sequential hardware trojan: side-channel aware design and placement. In: 2011 IEEE 29th International Conference on Computer Design (ICCD). IEEE (2011)
10. Lei, X., et al.: Modeling tabular data using conditional gan. *arXiv preprint* arXiv:1907.00503 (2019)