Received 6 April 2023, accepted 26 April 2023, date of publication 1 May 2023, date of current version 16 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3272034

RESEARCH ARTICLE

Detecting Unknown Hardware Trojans in Register Transfer Level Leveraging Verilog Conditional Branching Features

SARWONO SUTIKNO[1], **(Member, IEEE), SEPTAFIANSYAH DWI PUTRA**[2], **FAJAR WIJITRISNANTO**[1,3], **AND MUHAMAD ERZA AMINANTO**[4], **(Member, IEEE)**
[1]School of Electrical Engineering and Informatics, Bandung Institute of Technology, Bandung, West Java 40132, Indonesia
[2]Lampung State Polytechnic, Bandar Lampung, Lampung 35142, Indonesia
[3]National Cyber and Crypto Agency, Depok, West Java 16518, Indonesia
[4]Cyber Security Program, Monash University Indonesia, Bumi Serpong Damai 12150, Indonesia

Corresponding author: Sarwono Sutikno (ssarwono@stei.itb.ac.id)

This work was supported by the Research, Community Service, and Innovation Program, Bandung Institute of Technology (P3MI-ITB).

ABSTRACT Hardware Trojans have concealed modifications to integrated circuits (ICs) that can alter their functions, performance, or security properties. Existing Trojan detection methods are designed primarily to detect Trojans at the gate-level IC abstraction and lower levels, and only a few studies have investigated Trojan detection at the register transfer level (RTL). This study presents a novel machine learning-based approach for RTL-level Trojan detection, which leverages conditional statements from Verilog/VHDL code as ML features. Our proposed method has several significant novelties. Firstly, it can detect unknown Trojan instances since we incorporate general features for all Verilog circuits. Second, our approach can detect nontrigger-based Trojans, which are a type of Trojan that is particularly challenging to detect and has not been addressed by many existing techniques. Third, we incorporate feature engineering techniques, including Mutual Information and Person-Coefficient Correlation, to select the best features. We also implement feature scaling with standardization before balancing the data set. Our experimental results demonstrate that our approach achieves an average accuracy of 95.65% in the detection of Trojans, which is higher than previous detection techniques. The method is tested in the Trust-Hub Trojan benchmark RTL design, which demonstrates its effectiveness in detecting a wider range of Trojans at the RTL level. In summary, our novel approach shows great promise for enhancing hardware security by detecting a wider range of Trojans, including previously unseen Trojans, and improving detectionÂ accuracy.

INDEX TERMS Hardware trojan, RTL, machine learning, detection, Verilog, VHDL.

I. INTRODUCTION

It is a common practice in the semiconductor industry to use third-party intellectual property (3PIP) to meet the specification requirements in IC design. This practice is due to the high production demands and design complexity that must be met in a limited time. The integration of various intellectual properties (IP) makes various parties involved in most of the processes, both during design, fabrication, and assembly [1]. They have the opportunity to make additions or changes to the design logic for malicious purposes, which are also known as hardware trojans. The resulting impact varies from denial of service (DoS), decreased performance, and changes in function, to leakage of confidential information [2].

The numerous uses of IC in various fields such as the automated transportation industry, communication, health, and military make the threat of hardware trojans important to overcome [3]. As an example in the military sector, the attack by hardware trojans on weapon systems can escalate from small to systemic impacts. Trojans can cause a failure to a single subsystem of a weapon system or a systemic failure to a whole system, especially in Network Centric Operations

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Liu.

(NCO) [4]. Several strategies were eventually developed to overcome trojans in the form of hardware trojan detection, design for security (DfS), bus security, and secure architecture [5]. Trojan detection is a widely developed technique in research [6], [7]. This trojan detection technique is included in the trojan mitigation at the pre-silicon stage.

Hardware trojan detection techniques are further developed by involving ML in the process [8]. The main reason for this participation is none other than the increase in computing capabilities to support the application of ML in various fields [9]. The beginning of ML integration into the trojan detection technique was carried out in 2015 [10]. Detection strategies using ML include reverse engineering, circuit feature analysis, and side-channel features [5]. Circuit feature analysis is a superior solution, as reverse engineering and side channel require golden IC for comparison [11].

The implementation of circuit feature analysis techniques to design the ML detector trojan model is carried out in various ways. ML can be used as a trojan classifier based on data from switching activities and netlist features [12]. Less-toggled signal (LTS) measurement data in gate-level abstraction can also be used as a basis for ML design [13]. Several studies have adopted ML for detection by exploiting gate-level abstraction functional and structural features [14], [15]. Some studies can even use circuit testability numbers to produce high-performance models for gate-level detection [16]. Although many implementations of ML have been carried out, the implementation for detection still revolves around gate-level abstractions and other low-level abstractions. Meanwhile, the RTL abstraction, commonly used in the design stage, has the same vulnerability to hardware trojans.

Although many hardware trojan detection techniques have been proposed in RTL, most methods do not involve ML in the research mechanism. For example, registration detection based on the no-data corruption feature was developed without ML [17]. Another technique based on control flow subgraph matching has been carried out successfully, although it relies on a library containing trojan variation [18]. The technique was also developed using a probabilistic neural network, even with the disadvantage of using high computational resources [19]. Another proposal that uses the abstract syntax tree (AST) has been successfully developed with good performance, but for now it is applied in a small core coverage area and is limited to the XGBoost single ML model [20]. There is also a framework (nonmodel) that has been developed to determine the location of the RTL logic that has the potential to have a Trojan function [21]. Of these many studies, only Choo et al.'s [22] research integrated detection techniques with ML, which has a reasonably broad IP core coverage. Choo et al.'s study [22] uses a variety of ML models that use the features of the RTL code for the execution probability of conditional statements. Although it has achieved an excellent average accuracy of 91.91%, this technique has several shortcomings that can be improved, such as overlap of feature characteristics, difficulty in extracting features, accuracy, detection coverage, and limitations of the ML algorithms involved.

Taking into account the shortcomings related to ML integration in hardware trojan detection in RTL, it is necessary to overcome the deficiencies found in previous studies. This manuscript proposes an improvement over Choo et al.'s [22] which is ability to detect non-trigger bases trojans for obtaining a trojan detection model with high average accuracy performance, wide IP core coverage, and a detection mechanism with efficient and practical features. In addition, the involvement of various ML models that require efficient computing resources is also considered in the model's design. This study also addresses a significant limitation of the previous study [23], namely the inability to detect zero-day trojans. In other words, the proposed method can detect unknown trojans that do not exist in the training data set. In general, the manuscript contributions are threefold as follows.

- The capability to detect new (unknown) trojans that are not available in the training dataset. We incorporate parameter features that are general for all Verilog circuit codes, not only particular codes.
- The proposed model could detect a non-trigger-based trojan, which cannot be detected by Choo et al. [22]. Therefore, this study covers broader sets of trojans, including the trigger-based and non-trigger-based trojans.
- A novel extracted feature in which we exploit conditional parameters in bit units of code branches. Choo et al. [22] used the execution probability of code branches instead. We also leverage features engineering using Mutual Information and Person Coefficient Correlation to define the best features. This technique leads to higher accuracy in the detection capability of the trained model that reaches 95.65%.

The remainder of this paper begins with an explanation of definitions and the relevant works in Section II. Section III discusses the research methodology for designing ML models based on Trust-Hub benchmark data. The main discussion is described in Section IV, which elaborates performance evaluation and comparison with previous research. The article is concluded with Section V, consisting of conclusions and future research opportunities.

II. DEFINITIONS AND RELATED WORKS

Research related to hardware trojans began in 2007. Various suggestions have been made, starting with creating a trojan design based on multiple triggers and payloads. On the other hand, design for trust (DfT), split manufacturing, and trojan detection are formulated as trojan handling methods [7]. The research trend then continues by studying the analysis of trojan attacks and new mitigation techniques, such as the vulnerability analysis of trojan injection in the circuit [21], [24]. The trojan benchmark became a hot topic as an objective comparison solution of all existing mitigation techniques [25].

Furthermore, integrating ML into trojan detection techniques has become the subject of much research [5].

A. HARDWARE TROJAN DETECTION

The design, fabrication and distribution complexity of electronic devices has forced the collaboration of IC design and the manufacturing industry to create the desired core chip. As a result, more and more untrusted 3PIPs are involved in the supply chain. Increases the risk of hardware trojan injection.

These hardware Trojans can be added at various levels of IC design abstraction, both at RTL abstraction, firm IP (gate-level), and complex IP (physical). The chances for pre-silicon (before hard IP) attacks appear to be fairly balanced between the various abstractions. Therefore, the disparity of trojan detection solutions between fewer RTL abstractions and mature gate levels must be overcome by developing new solutions for RTL trojan detection [1].

A trojan benchmark is used as the source of the dataset to build an RTL trojan detection model. The benchmark used is Trust-Hub, which is designed according to hardware trojan taxonomy to make it easier for researchers to focus on their individual research needs [25]. The taxonomy shows the different benchmarks available on the Trust-Hub platform [1]. In this paper, the focus of the dataset is taken from the benchmark based on the RTL abstraction taxonomy.

Research on RTL trojan detection is started later than work on the gate level. Therefore, some studies have not yet integrated cross-scientific techniques such as ML in detection. This practice is common in gate-level netlists. However, some of these studies have used the Trust-Hub benchmark based on the RTL taxonomy as the primary research data. Several studies that have not yet implemented ML but use the Trust-Hub benchmark are the no-data corruption feature-based technique by Rajendran et al. [17], the control flow subgraph matching technique by Piccolboni et al. [18], and the RTL vulnerability framework by Islam et al. [21].

The method of Rajendran et al. [17] intends to complement the previous detection technique. Modifications to the Trust-Hub design were made to have specific registers as conditions for the attack. Furthermore, a method is designed that successfully detects trojans in the registered design made on the AES core, except for AES-T1200. Although successful, this technique is inefficient because it is based on automatic test pattern generation (ATPG) and bounded model checking (BMC), which are NP-complete problems. Moreover, the injection and the benchmark methods have been modified, and thus they cannot be objectively compared with other Trust-Hub benchmark users.

The method of Piccolboni et al. [18] provides automated verification techniques to detect trojans in RTL designs. The characteristics of the trojan are taken from the control flow. The control flow uses a graph on the trigger and payload of the trojan. Then, these characteristics are collected in the library as a basis for detection reference. The study used the Trust-Hub benchmark, which claims that the technique could detect most trojans. However, this method's high false positive (*FP*) value and dependence on library updates are significant drawbacks.

Islam et al. [21] empirically and analytically analyze the transition activity in crypto-based cores, digital signal processing (DSP) and macro-blocks (adders and multipliers). The dual-bit-type (DBT) model is used to measure the sparse activity of the signal based on the input. This technique is claimed to determine the statistical relationship between the input and the circuit's sparse conditions and to provide the position of the sparse conditions. The rare condition in the form of a trojan trigger could be detected in a short time without requiring simulation. Therefore, it is concluded that trojans with no-trigger characteristics are excluded from the context of this study.

B. MACHINE LEARNING FOR TROJAN DETECTION

An ML-based technique for hardware trojan detection in RTL abstraction was developed to determine the limitations of the previous technique. Some proposed techniques are developed from scratch, while others are developed by adding ML to an existing approach. Both supervised and unsupervised ML algorithms have been involved in supporting hardware trojan detection techniques. Supervised learning uses labeling in the training data set to conduct training in the built model. Meanwhile, unsupervised learning tries to find the relationship between data by looking at the characteristics of each data without requiring a label from the training dataset. Some supervised ML algorithms include the support vector machine (SVM), Naïve Bayes, Logistic Regression, Decision Tree (DT) and K-Nearest Neighbors (K-NN). Meanwhile, the unsupervised algorithms are K-means Clustering and Density-Based Spatial Clustering (DBSCAN).

Most ML-based RTL-level trojan detection research uses supervised learning algorithms in its approach. In 2017 Demorozi et al. [19] improved the Piccolboni et al. detection technique, previously involving the probabilistic neural network (PNN) model of ML. Modifications were also made regarding the detection model approach through graph isomorphism-based verification of the RTL trojan characteristics. The detection results show the ability to recognize trojans on several Trust-Hub benchmarks such as AES-T400, AES-T600, AES-T700, RS232-T200, RS232-T600, RS232-T901 and all Basic RSA trojans. The coverage of the benchmark detection is less than the reference technique of Piccolboni et al. [18]. However, the characteristics of the neural network that are easily adapted to new datasets with high computational resource costs make this model superior.

Han et al. [20] in 2019, also designed a hardware trojan detection model in RTL abstraction. This detection model utilizes features extracted from the abstract syntax tree (AST). The evaluation results show that the model can simultaneously detect AES core RTL trojans as many as 21 benchmarks. The training uses a Gradient Boosting algorithm with the XGBoost library. Conceptually, this feature-based AST model can be further developed for the characteristics of the related training data set to allow the detection of a larger data set. However, this model has limitations on one type of gradient-boosting algorithm for ML and has not explored the possibility of other types of ML algorithms.

Choo et al. [22] in 2020 developed a trojan detection model for RTL abstraction based on the features of the Verilog/VHDL code. These features are obtained from conditional statements such as if, elsif, and case, which are then defined as branching features of the code. Various features are proposed, namely Branching Probability P, Outer level Branching Probability P_{outer}, Effective Branching Probability P_e and Relative Branching Probability RP. The extraction of each feature is done by calculating the probability that a conditional statement will be executed or not when the implementation or synthesis of the RTL code is implemented. After utilizing several algorithms for the learning process, namely Logistic Regression, Decision Tree, k-nearest neighbor, and SVM, satisfactory results were obtained for the detection capabilities of the Trust-Hub core AES and wbconmax benchmarks. The evaluation metric shows a valid positive rate (TPR) of 93% and an accuracy of 91.91%. However, several model components have the potential to be improved to obtain better evaluation metrics, such as the explanation in the Introduction.

III. METHODOLOGY

In this study, we proposed a trojan detection that contains four modules as shown in Figure 1. We prepare the data and extract the features in preprocessing module that discussed in sub-section A and B. While the rest of modules are explained in the sub-section C and D. We use the RTL trojan hardware sample from the Trust-Hub benchmark. Trust-Hub benchmark sourced from https://www.trust-hub.org/ taxonomy was chosen because researchers trusted it to train trojan classification models. RTL trojan samples used include core AES, wbconmax, RS232, and BasicRSA. Intellectual property (IP) in the form of these cores was chosen so that the built solution can detect various characteristics of trojans from different sets of cores whose scope is broader than the reference model proposed by Choo et al. [22].

A. DATA PREPARATION

The data preparation stage includes four things, namely data integration, data cleaning, data transformation, and data reduction. Data integration includes feature design, feature extraction per sample, and merging samples from these various core IP benchmarks into a single dataset. Data cleaning is intended to check the existence of data with empty values or outliers. The data set then undergoes a transformation process to change the scale of features with an overvalued range of values. The next stage is the reduction or modification carried out in the form of balancing the population of the training data class. The new feature design phase (feature engineering) is discussed in detail in the Discussion section. Meanwhile, the feature extraction process from the RTL description was carried out with a procedure similar to Choo et al.'s P, P_{outer}, P_e, RP, and C_{dep} feature extraction. For example, the process of extracting one of the features, namely P or Branching Opportunity, is shown in Figure 2.

The fork probability P is defined as the probability of executing a Verilog/VHDL code branch. According to Figure 2, there is one block of code (lines 63 to 69) marked with each always (or assign) statement that contains two branches of code in the form of an if-else conditional statement. In branch number 1, line 65, the if statement is known to require a signal condition of $rst = 1$. Meanwhile, on the 67th line, number 2, there is an else if statement which requires the condition of register $Tj_{Trig} = 1$. The first branch can be executed when the 1-bit rst signal is active. Then the probability of branching number 1 is $P_{1.1} = \frac{1}{2}$. While branch number 2, apart from having a 1-bit Tj_{Trig} signal condition, is also affected by the 1-bit rst condition in the first branch. So, the probability of executing branch number 2 follows the signal conditions $rst = 0$ and $Tj_{Trig} = 1$, resulting in $P_{1.2} = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$. Although there are only two code branches explicitly, there is one conditional statement that has not been involved, namely with signal conditions $rst = 0$ & $Tj_{Trig} = 0$ which results in $P_{1.3} = \frac{1}{4}$. Thus, the total probability of branching in the block in Figure 2 is $P_1 = P_{1.1} + P_{1.2} + P_{1.3} = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$. The branching probability is therefore calculated according to Equation 1, with $|COND|$ being the number of possible values that can activate the if-else statement and N_{cond} being the number of bits required in the conditional statement.

$$P = \frac{|COND|}{2^{N_{cond}}} \tag{1}$$

In this paper, the proposed ML models for trojan detection are Logistic Regression, Decision Tree, SVM, KNN, Naïve Bayes, Random Forest, and Gradient Boosting. The use of supervised learning algorithms for RTL trojan detection chosen in this paper results in the need to label each feature set for each sample. Labels are assigned to each obtained sample based on the position of the if, else, elif, or assign conditional statements in the Verilog/VHDL code. The function of the conditional statement is called the code branch. Thus, each code branch has two possible labels: trojan and non-trojan.

Labeling of the samples is done as follows. When an if, else, elif, or assign statement is found, it is identified whether the statement activates the wire/variable $Tj_{Trig} = 1$ or not. When an if, else, elsif, or assign statement is found and the branching of the trojan code. If true, then the branching code is labeled trojan. This branch of the code is also classified as a trojan.

Each feature and label obtained from the static code analysis are then fed into the ML model. The feature in question is the result of feature engineering, whose mechanism is explained further in the Modeling Chapter. The entire sample then underwent a data set diversification process, which was calculated based on the total number of data sets before being used in the ML process. The diversification process in question is that the entire sample is divided into three data sets with the amount of allocation according to best practice, namely training, validation, and test sets.

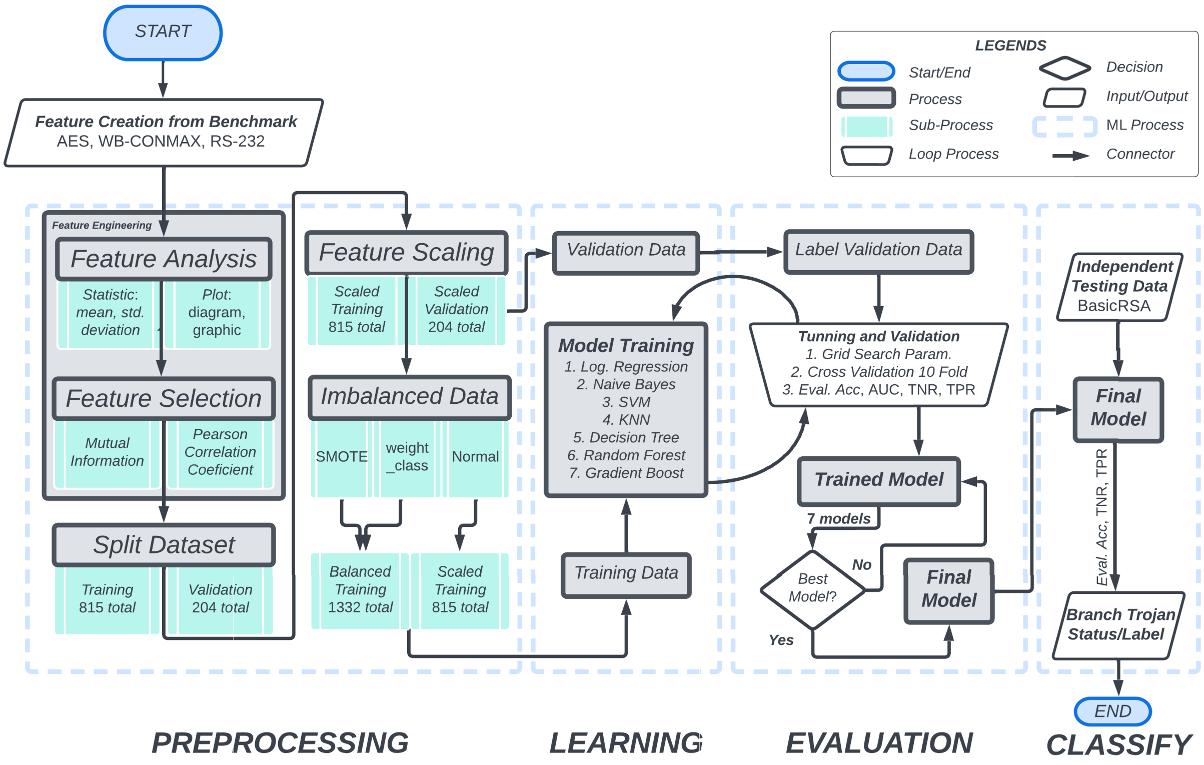
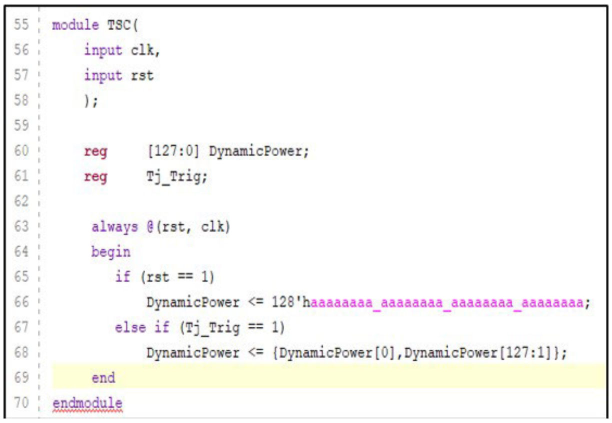


FIGURE 1. The proposed method overview with four modules: preprocessing, learning, evaluation and classify.


```
module TSC(
    input clk,
    input rst
    );

    reg     [127:0] DynamicPower;
    reg     Tj_Trig;

     always @(rst, clk)
     begin
        if (rst == 1)
            DynamicPower <= 128'haaaaaaaa_aaaaaaaa_aaaaaaaa_aaaaaaaa;
        else if (Tj_Trig == 1)
            DynamicPower <= {DynamicPower[0],DynamicPower[127:1]};
     end
endmodule
```

FIGURE 2. Sample benchmark AES-T500.

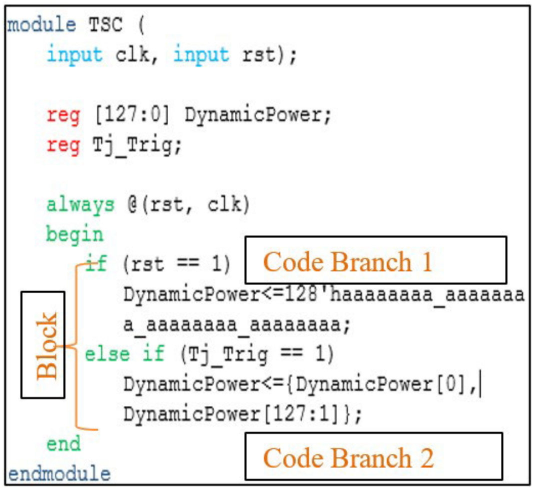
The core AES, wb-conmax, and RS232 benchmark datasets were used in this study for training and validation sets. The training set is used to train the model built by entering a sample in the form of branching code. The three IP cores are then divided proportionally for the training set of 80% of the code branching population and the remaining 20% for the validation set. The validation set is used to validate the model hypotheses used in the training set to see the performance of the new data and the basis for improving the model's performance. In total, 1019 code branching samples were extracted from the AES, wb-conmax, and RS232 benchmark cores for the training and validation sets. The sample, if grouped according to the number of core benchmarks involved, reached 33 benchmarks with details of 21 AES benchmarks, two wb-conmax benchmarks, and 10 RS232 benchmarks. In addition to the training and validation sets, the BasicRSA benchmark dataset is also used as a test set. The test set is used to objectively test the performance of the detection model using an independent dataset that is not involved in the training or validation set. In this basic RSA data set, there are four benchmarks that, when extracted, yield 184 code branching samples.

B. FEATURE ENGINEERING

The P, P_{outer}, P_e, RP, and C_{dep} features proposed by Choo et al. do not represent the relationship between code ramifications or between different code blocks [21]. Therefore, this study engineered new features to overcome these problems while maintaining the representation of code branch execution opportunities in the previous feature. In addition, the new features are designed to have an easy extraction process as a substitute for the relatively complicated feature extraction of Choo et al. Some new features can be built using the characteristics of both code branching and code blocks. The methodology explains that a code block can have multiple code branches. The new features are thus structured, considering the size of the conditional bit and the type of conditional statement. Figure 3 is used to describe the feature engineering process. In Branch Codes 1 and 2, for example, the conditional bit size of each signal rst and Tj_{Trig} is 1. Meanwhile, each branch has if and other if statements when viewed from a different perspective. Several new code branching features

TABLE 1. New code branching proposed features.

Features	Representation	Value at Branching Code 1
Branchbit	The size of the conditional bits for each code branch	$\|rst\| = 1$
Blockbit	Total conditional bits for all code branches in a block	$\|rst\| + \|Tj_{Trig}\| = 2$
MaxBranchbit	The largest conditional bit size among all code branches in a block	$\|rst\| = \|Tj_{Trig}\| = 1$
MinBranchbit	The smallest conditional bit size among all code branches in a block	$\|rst\| = \|Tj_{Trig}\| = 1$
Blockbranch	Number of code branches in a block	2
Blockbranch	Conditional statement	if



```
module TSC (
   input clk, input rst);

   reg [127:0] DynamicPower;
   reg Tj_Trig;

   always @(rst, clk)
   begin
     if (rst == 1)
        DynamicPower<=128'haaaaaaaa_aaaaaaa
        a_aaaaaaaa_aaaaaaaa;
     else if (Tj_Trig == 1)
        DynamicPower<={DynamicPower[0],
        DynamicPower[127:1]};
   end
endmodule
```

FIGURE 3. Verilog code block example.

can be formulated using these two characters, as shown in Table 1.

The relationship between code blocks based on code branching in Table 1 is realized by the Blockbit, MaxBranchbit, and MinBranchbit features. All three compare the conditional bit size of each branch in one block with other code block conditions. Branchbit, on the other hand, describes the unique character of each code branch so that it can be used to compare code branches. The blockbranch describes the relationship of the number of branching codes owned by a block. Meanwhile, the statements explain the relationship between code branches by representing their respective conditional statements. The six proposed new features were then verified for their association with the trojan label using Mutual Information (MI) and Pearson's correlation. The goal is to find out how much information can be obtained from each feature to determine the trojan label. Features that contribute little to determining the trojan label will be removed from the list of proposed features.

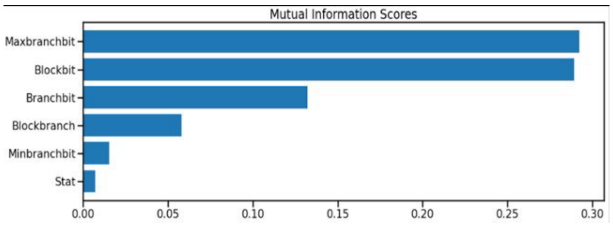


FIGURE 4. Mutual information score on proposed features.

$$I(X;X) = H(X) - H(X|Y) \tag{2}$$

The MI value was chosen because it represents the level of dependency between two random variables (features and labels). The MI between two random variables X and Y is expressed according to Equation 2 where $I(X;Y)$ is Mutual Information for X and Y. Meanwhile, $H(X)$ is the entropy value for X and $H(X|Y)$ is the conditional entropy for X if known Y. Entropy itself is a description of how much information there is in a random variable. This is related to the probability of the occurrence of the corresponding variable. A low probability of occurrence results in the related variable having more information than the variable with a high probability of occurrence. For example, a variable X that has a probability of occurrence $p(X)$ with an information entropy of $H(X) = -log(p(X))$ will have a final entropy of $-\Sigma p(X)log(p(X))$.

Calculating the MI value for each proposed feature produces various values with MI MaxBranchbit 0.292096, Blockbit with MI 0.289252, Branchbit with MI 0.132256, Blockbranch with MI 0.058178, MinBranchbit with MI 0.015447, and Statement with MI 0.007219. Figure 4 shows a histogram of comparisons of MI scores for each characteristic. Low scores are known to be owned by MinBranchbit and Statement features.

After knowing the MI value, the next step is to calculate the correlation between features using Pearson's correlation. The Pearson correlation coefficient calculates the significance of the connection or linear relationship between two variables. The Pearson correlation coefficient between two variables x and y in the i-th sample of a data set is formulated according to Equation 3. The values of x and y are the average values of each variable (feature) in all data sets involved. The correlation coefficient has a range of values from -1 to 1. Each describes a perfect correlation relationship negatively and positively. The closer the zero is, the more unrelated the two variables will be.

$$\rho = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2 \sum_i (y_i - \overline{y})^2}} \tag{3}$$

In Figure 5, the results of the correlation calculation between the features of the new proposal are shown to determine the relationship of each feature with each other. It can be seen that the Blockbranch and Statement features each

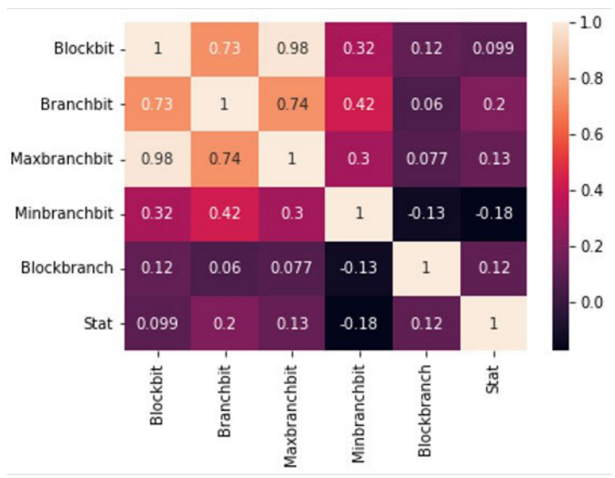


FIGURE 5. Pearson's correlation score on proposed features.

have a relatively low correlation when associated with other features. Based on the MI and the correlation coefficient results, features with a fairly low MI value and correlation, namely Statements, are excluded from the new feature candidates, so that only five new code branching features are used. The Statement feature is excluded from the feature list because it has an MI value below the 0.01 threshold with the highest correlation coefficient, which is below the 0.4 threshold. Blockbranch, even though it has a similar correlation value, has a fairly large MI value of 0.058178. Statement deletion was also decided because the categorical Statement feature has different characteristics from other features that are continuous. Five selected features, Blockbit, MaxBranchbit, MinBranchbit, Branchbit, and Blockbranch, are then used in the model design stage.

C. MODELING

When viewed in detail, this study's type of ML algorithm is a superset of the algorithm used by Choo et al.'s reference paper. The addition of the Naïve Bayes algorithm, which has not been carried out by Choo et al. opens the opportunity to obtain a choice of models with better performance. All of these models require various stages of repeated trials to achieve maximum performance. The model built in this paper applies several stages of iterative modification and improvement analysis, as shown in Figure 1. The early-stage model of this modeling technique was published earlier and also influenced this research [23].

Feature selection is performed using various feature engineering operations. In the preprocessing stage, the relevant features are engineered and extracted from the code branching sample to be used as a class-distinguishing parameter in the classification case. This is done to obtain features well related to the dataset label status. The data set is further divided into training data sets and validation data sets. Data modification processes such as standardization and oversampling are carried out at this stage to ensure that the data set has balanced characteristics for the entire sample. The learning algorithms are selected and trained to generate models from

TABLE 2. Feature extraction results of code branching samples.

Benchmark	Blb	Brb	Mxb	Mnb	Bbr	Trojan
AES-T100	20	20	20	20	1	0
	4	4	4	4	1	1
	1	1	1	1	2	0
	1	1	1	1	2	0
...	...	...	...	...	...	...
WB-T200	32	32	32	32	2	1
	32	32	32	32	2	1
	2	2	2	2	2	0
	2	2	2	2	2	0
	...	...	...	...	...	...
	1	1	1	1	1	0
...	...	...	...	...	...	...
RS232-T901	8	8	8	8	2	1
	8	8	8	8	2	1
	2	1	1	1	2	0
	2	1	1	1	2	0

the training data set in the learning stage. The resulting model is expected to be consistent with the character of the training data. After that, various processes such as cross-validation and evaluation of the results were carried out to determine the final model. The final model is utilized in the classification stage to evaluate the model with entirely new data. At the evaluation stage, the final model produced is then tested using a test dataset to evaluate its performance using evaluation metrics. The entire design of the ML model used in this research was built with Jupyter Notebook 6.4.8 on the Anconda3 platform. Several libraries are sklearn, py-xgboost, pandas, numpy, matplotlib, and seaborn. Furthermore, the Verilog/VHDL code feature engineering process is performed using Vivado Xilinx 2020.2, RapidMiner, and ModelSim PE Starter Edition for both code feature analysis and simulation. All tools run on a local PC with 8 GB RAM and AMD Ryzen 5 Processor.

D. MODEL TRAINING

Feature extraction was performed on Trust-Hub AES, wb-conmax, and RS232 benchmark data sets as a data preparation for model training. The three benchmarks yielded at least 1019 total samples. The feature extraction process for each sample is carried out by analyzing static code in each Verilog file from the benchmark. The calculation results of the Blockbit (Blb), MaxBranchbit (Mxb), MinBranchbit (Mnb), Branchbit (Brb) and Blockbranch (Bbr) features for each code branch sample of all benchmark modules are shown in Table 2. Each row in Table 2 represents a sample branch of code indicated by a specific conditional statement. Up to 80% of the 1019 samples were randomly assigned to the training data set. In comparison, the remaining 20% are used for the validation of the data set. The datasets thus have a number close to 815 and 204 code branching samples. It is known that the 1019 samples involved do not include the results of the Basic RSA sample extraction. This is because the branching sample of the Basic RSA code will be used after the training process as a test data set.

The collected feature data set goes through a transformation process, namely feature scaling. Feature scaling is

TABLE 3. Training set sample standardization results.

Sample	Blb	Brb	Mxb	Mnb	Bbr
1	-0.31	-0.22	-0.29	-0.18	2.12
2	3.43	4.82	3.58	9.86	-0.72
3	-0.25	-0.26	-0.23	-0.18	0.22
...	...	...	...	...	...
813	-0.25	-0.14	-0.23	-0.18	0.22
814	3.55	4.86	3.61	-0.18	1.16
815	3.55	-0.26	3.61	-0.18	1.16

done using the standardization technique. Standardization is a procedure for changing the scale of features to have a value distribution close to the normal distribution. This functionality makes standardization a good solution for dealing with outliers. This standardization process is carried out using the StandardScaler module of the Sklearn model selection library for both training and validation datasets. An example of the results of the standardization of the training set is shown in Table 3. It can be seen that the value of each feature is in a much smaller range than the initial value of the feature.

After the standardization is completed, the next step is to overcome the problem of unbalanced training set samples. From 815 training set samples, 149 trojan samples were obtained compared to 666 non-trojan samples. An oversampling approach is used to handle this imbalanced dataset, with techniques such as SMOTE (Synthetic Minority Oversampling) and random oversampling. Each ML algorithm involved in the training process will experience a different approach related to the method used. This is done to optimize the model to obtain satisfactory AUC and accuracy.

The SMOTE technique is a method for synthesizing minority classes (trojan classes) from available samples so that class population balance can be achieved. In simple terms, SMOTE utilizes the KNN model to determine a new sample with neighbors taken as the sample with the highest frequency relative to the position of the new sample.

The next training stage is carried out after all datasets have been prepared. The construction of the trojan detection model for each algorithm is carried out using a k-fold cross-validation approach with k = 10. This is done so that the performance of the training model can be validly evaluated by minimizing the chance that the evaluation results are due to chance. The purpose of tuning is to find the best training parameters that produce a model with good accuracy against the validation data set. Using the k-fold cross-validation method, hyperparameter tuning is also applied to each model according to the characteristics of each model. This tuning is applied after the training process on the training dataset is carried out. This hyperparameter tuning process is implemented with the Jupyter Notebook tools utilizing the GridSearchCV module of the Sklearn model. This process produces a model with detailed parameters, as shown in Table 4.

E. EVALUATION

The performance evaluation of the ML model is carried out using the hardware trojan classification accuracy metrics, AUC (Area Under the ROC Curve), *TPR*/Recall, *TNR*, and Precision. All evaluation metrics refer to the concept of a confusion matrix.

TABLE 4. Model parameters of a 10-Fold cross-validation training process.

Model	Parameter
Logistic Regression	C=0.55; solver=newton-cg
Naïve Bayes	algor=MultinomialNB; alpha=0.1
KNN	C=0.55; algor=balltree; nneighbors=8; dist=Manhattan
SVM	C=0.15; kernel=rbf
Decision Tree	maxdepth=10; impurity(criterion)=gini; ccpalpha=0.00175; splitter=best
Random Forest	criterion=gini; maxdepth=5; maxfeatures=auto; nestimators=30
Gradient Boosting	criterion=gini; maxdepth=3; nestimator=100; learningrate=0.1

The confusion matrix describes the relationship between the predictions of the model and the condition or status of the predicted object labels. The true positive (*TP*) is output when the model correctly predicts a positive class (trojan). True negative (*TN*) applies equally to the case of class negative (non-trojan). False positive (*FP*), on the other hand, is a condition in which the model incorrectly predicts a positive class. Meanwhile, false negatives apply the same in the context of negative classes. Based on this definition, the accuracy value can be formulated according to Equation 4, which is the ratio of the number of correctly classified samples compared to the total number of all samples. In addition to precision, there are also *TPR* (Recall/Sensitivity) and *TNR* (Specificity) according to Equation 5 and Equation 6. There is also a precision metric formulated with $TP/(TP + FP)$ to determine how accurately a model predicts the status of trojans.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$TNR = \frac{TN}{TN + FP} \tag{6}$$

IV. PERFORMANCE EVALUATION, ANALYSIS, AND COMPARATION

A. MODEL EVALUATION

Based on the modeling results of the seven algorithms, several performance evaluation metrics are calculated, which are then used to determine the best model. The results of the calculation of the evaluation metric for the validation data set are shown in Table 5. The best model was selected. Then its performance was tested using the same evaluation metric as the test dataset, namely Basic RSA. This step was taken to verify that the best ML model is not only limited to being applied to training and validation datasets with branching features of the AES, wb-conmax, and RS232 code.

The best model is selected based on the best accuracy, *TPR* / recall, precision, and AUC performance against the

TABLE 5. Model evaluation performance on the validation dataset.

Model	Accuracy	AUC	Precision	Recall (TPR)
Logistic Regression	93.63%	95.85%	85.71%	72.73%
Naïve Bayes	91.18%	83.68%	74.20%	69.70%
KNN	96.57%	97.43%	100.0%	78.79%
SVM	89.32%	96.21%	82.76%	72.73%
Decision Tree	96.57%	97.51%	100.0%	78.79%
Random Forest	96.08%	97.09%	100.0%	75.76%
Gradient Boosting	96.57%	96.91%	100.0%	78.79%

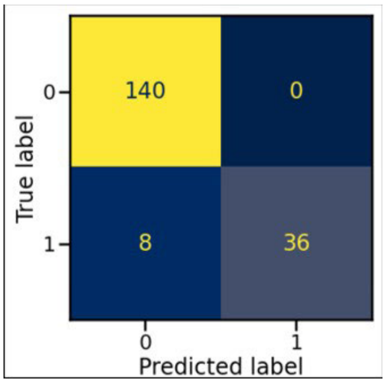


FIGURE 6. Confusion matrix RSA basic trojan classification.

validation of the data set. The model's performance with light-colored columns indicates the evaluation metric with the lowest score, while the dark color indicates the evaluation metric with the highest score. It can be seen that, in general, the Naïve Bayes model has the lowest performance, and conversely, the highest performance is owned by the Decision Tree algorithm followed by Gradient Boosting (Ensemble). Although Gradient Boosting has almost the same high performance, Decision Tree is more prioritized because ensemble learning computing resources tend to be higher than algorithms with a single tree structure such as Decision Tree. Decision Tree was thus chosen as the best ML model for trojan detection. To prove that the model can be applied to other unrecognized code branching datasets, the model is evaluated using the RSA Basic Benchmark sample. Although the sample RSA code branching has a different module name, it has the same code branching characteristics as the three previous benchmark datasets AES, wb-conmax, and RS232. The RSA-T100 to T400 Basic datasets go through a standardization process, as during the last training and validation datasets. The tests were then carried out on the Decision Tree with a maximum tree depth parameter of 10, an impurity calculation based on Gini, and a pruning complexity of 0.00175. The results of the metric evaluation show an accuracy value of 95.65%, Precision 100.0%, and TPR/Recall 81.81% as shown in the confusion matrix Figure 6.

Fig. 6, in the form of a confusion matrix, shows that the Decision Tree model can classify all Basic RSA trojans relatively well. It should be emphasized that the RSA Basic test data set is not involved in the training or validation process.

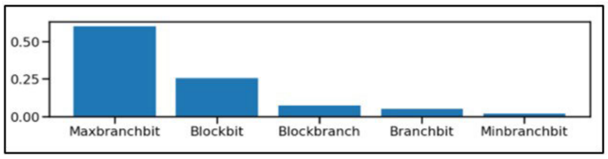


FIGURE 7. Feature importance model decision tree.

This approach is different from the research of Choo et al. where the training and validation data sets were obtained from the ADASYN synthesis of the test datasets, namely AES and the wb-conmax benchmark. As a result, there is the possibility of information leakage between the training data set and the test data set in the reference study. Therefore, it can be concluded that with the successful detection of trojans using the RSA Basic dataset, regardless of the training and validation set, the Decision Tree model in the paper has better validation.

B. MODEL ANALYSIS & COMPARISON

Decision Tree as the best model performs classifying trojan and non-trojan classes on dataset validation and testing using a particular tree structure. The structure has branch settings and feature value thresholds based on the Gini cost or impurity. The application of Gini impurity shows that the Decision Tree in the paper applies the CART (Classification and Regression Trees) algorithm. We believe that Decision Tree creates an ensemble of trees that have sparseness and different properties of the tree structure to isolate anomalies [26] instead of profiling common behaviors.

Each code branching feature thus has its level of importance, also known as feature importance. Feature importance is the assigning a score to each feature based on how significantly each feature predicts the target variable (trojan). This value is essential to measure in the Decision Tree model analysis to determine which features contribute the largest and the smallest in the classification process. In this paper model, the highest to lowest value of importance of characteristics is obtained sequentially by MaxBranchbit (0.747), MinBranchbit (0.078), Blockbit (0.074), Blockbranch (0.056) and Branchbit (0.044). The value is displayed in the histogram Fig. 7.

Therefore, the training, evaluation, and analysis results described prove that the Decision Tree model can perform well on training, validation, and testing data. The superiority of this model is one of several contributions made in this paper. Another contribution is that the characteristics of the paper model's training data set are broader than the reference model. The point of this statement is that the proposed model involves more types of trojans than the reference research model, which is characterized by the addition of AES, wb-conmax, and RS232 datasets, which are more heterogeneous than the reference model. Using the training dataset, the detection capacity of the Decision Tree is also proven to be wider than that of the reference model with the participation of Basic RSA. Table 6 comprehensively describes the com-

TABLE 6. Comparison of model detection coverage with other research.

Benchmark	Paper	[21]	[27]	[16]	[19]	[18]	[25]
AES T100	✓	✗	✗	✗	✓	✗	✗
AES T200	✓	✗	✗	✗	✓	✗	✗
AES T300	✓	✗	✗	✗	✓	✗	✗
AES T400	✓	✓	✗	✗	✓	✓	✓
AES T500	✓	✓	✗	✗	✓	✗	✓
AES T600	✓	✓	✗	✗	✓	✓	✓
AES T700	✓	✓	✗	✓	✓	✓	✓
AES T800	✓	✓	✗	✓	✓	✗	✓
AES T900	✓	✓	✓	✗	✓	✗	✓
AES T1000	✓	✓	✗	✗	✓	✓	✓
AES T1100	✓	✓	✗	✗	✓	✗	✓
AES T1200	✓	✓	✗	✗	✓	✗	✓
AES T1300	✓	✓	✗	✗	✓	✓	✓
AES T1400	✓	✓	✗	✗	✓	✗	✓
AES T1500	✓	✓	✗	✗	✓	✗	✓
AES T1600	✓	✓	✗	✗	✓	✗	✓
AES T1700	✓	✓	✗	✗	✓	✗	✓
AES T1800	✓	✓	✓	✗	✓	✓	✓
AES T1900	✓	✓	✓	✗	✓	✗	✓
AES T2000	✓	✓	✓	✗	✓	✗	✓
AES T2100	✓	✓	✗	✗	✓	✗	✓
wb-con T200	✓	✓	✗	✗	✗	✗	✗
wb-con T300	✓	✓	✗	✗	✗	✗	✗
RS232 T100	✓	✗	✗	✗	✗	✗	✓
RS232 T200	✓	✗	✗	✗	✗	✓	✓
RS232 T300	✓	✗	✗	✗	✗	✗	✓
RS232 T400	✓	✗	✗	✗	✗	✗	✓
RS232 T500	✓	✗	✓	✗	✗	✗	✓
RS232 T600	✓	✗	✗	✗	✗	✓	✓
RS232 T700	✓	✗	✓	✗	✗	✓	✓
RS232 T800	✓	✗	✗	✗	✗	✗	✓
RS232 T900	✓	✗	✗	✗	✗	✓	✓
RS232 T901	✓	✗	✗	✗	✗	✓	✓
BscRSA T100	✓	✗	✗	✗	✗	✓	✓
BscRSA T200	✓	✗	✗	✗	✗	✓	✓
BscRSA T300	✓	✗	✗	✗	✗	✓	✓
BscRSA T400	✓	✗	✗	✗	✗	✓	✓
B19 T300	✓	✗	✗	✗	✗	✗	✗
B19 T400	✓	✗	✗	✗	✗	✗	✗
B19 T500	✓	✗	✗	✗	✗	✗	✗
memctrl T100	✓	✗	✗	✗	✗	✗	✗
PIC16F84 T100	✗	✗	✗	✓	✗	✗	✗
PIC16F84 T200	✓	✗	✗	✗	✗	✗	✗
PIC16F84 T300	✓	✗	✗	✓	✗	✗	✗
PIC16F84 T400	✓	✗	✗	✓	✗	✗	✗
MC8051 T200	✓	✗	✗	✗	✗	✗	✗
MC8051 T300	✓	✗	✗	✗	✗	✗	✗
MC8051 T400	✓	✗	✗	✓	✗	✗	✗
MC8051 T500	✓	✗	✗	✗	✗	✗	✗
MC8051 T600	✗	✗	✗	✗	✗	✗	✗
MC8051 T700	✓	✗	✗	✓	✗	✗	✗
MC8051 T800	✓	✗	✗	✓	✗	✗	✗

parison of detection coverage between the model paper and related studies.

It can be concluded that this paper model has the better ability to detect IP hardware Trojans, regardless of the trigger conditions of the trojan, the functionalities and the types of IP cores that the trojan is targeting. However, there are limitations to the model that only claims to be able to handle RTL trojan detection based on Trust-Hub benchmark conditions. Furthermore, the characteristics of the AES, wb-conmax, RS232, and Basic RSA trojans have not been able to represent all possible triggers and functional trojans that are spread throughout the IC development supply chain. However, it should be emphasized that choosing an independent sample as a test data set and a detection coverage test data set is more objective than using a data set with characteristics similar to the training data set, as Choo et al. [22] did with AES and wb-conmax.

V. CONCLUSION

This research produced an ML-based RTL-level hardware trojan detection model using the Verilog/VHDL code branching features Blockbit, MaxBranchbit, MinBranchbit, Branchbit, and Blockbranch. The features are selected based on the effort to represent the relationship between branches and code blocks that prioritize the ease of feature extraction and the results of better accuracy and detection coverage. Furthermore, the selection of the ML model with a higher number of algorithms than the candidate model of Choo et al. also involved adding the Naïve Bayes algorithm, the Random Forest, and the Gradient Boosting. These treatments aimed to improve the Choo et al. ML model. The results showed that Decision Tree was chosen as the best model with a precision of 96.57% and AUC 97.51% against the validation set, which was the benchmark sample of AES, RS232, and wb-conmax. The Decision Tree model can achieve 95.65% accuracy, 100.0% precision, and 81.81% *TPR*/Recall in classifying independent data on the Basic RSA benchmark. This accuracy value is better than Choo et al.'s 91.91% as a comparison model that tested the AES and wb-conmax datasets. It is also proven that the proposed model can predict new data for a wide range of cores. Although satisfactory results have been achieved, there are some limitations to this study. It is necessary to improve the model to maximize the *TPR*/Recall value, which is generally lower than accuracy and precision. This modification should be done without reducing accuracy or *TPR*. In addition, it is possible to involve a neural network model so that it is assumed to obtain better evaluation metric results, of course at the cost of high computational resources.

REFERENCES

[1] S. Bhunia and M. Tehranipoor, *Hardware Security: A Hands-On Learning Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2018.

[2] M. Tehranipoor, "New directions in hardware security," in *Proc. 29th Int. Conf. VLSI Design 15th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2016, pp. 50–52.

[3] Y.-Q. Lv, Q. Zhou, Y.-C. Cai, and G. Qu, "Trusted integrated circuits: The problem and challenges," *J. Comput. Sci. Technol.*, vol. 29, no. 5, pp. 918–928, Sep. 2014.

[4] A. Sumari and S. Sutikno, "Cyber-physical systems threats, risks, and vulnerabilities: A challenge to Indonesia defense sector," in *Proc. 3rd Indonesia Int. Defense Sci. Seminar (IIDSS)*, 2019, pp. 347–364.

[5] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware Trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10796–10826, 2020.

[6] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Proc. IEEE Int. Workshop Hardw.-Oriented Secur. Trust*, Jun. 2008, pp. 15–19.

[7] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons learned after one decade of research," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1–23, Dec. 2016.

[8] Y. Jin, D. Maliuk, and Y. Makris, "Post-deployment trust evaluation in wireless cryptographic ICs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 965–970.

[9] R. Elnaggar and K. Chakrabarty, "Machine learning for hardware security: Opportunities and risks," *J. Electron. Test.*, vol. 34, no. 2, pp. 183–201, Apr. 2018.

[10] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, N. Sklavos, and F. C. Plessas, "Conventional and machine learning approaches as countermeasures against hardware Trojan attacks," *Microprocessors Microsyst.*, vol. 79, Nov. 2020, Art. no. 103295.

[11] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proc. IEEE*, vol. 102, no. 8, pp. 1229–1247, Aug. 2014.

[12] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[13] E.-R. Zhou, S.-Q. Li, J.-H. Chen, L. Ni, Z.-X. Zhao, and J. Li, "A novel detection method for hardware Trojan in third party IP cores," in *Proc. Int. Conf. Inf. Syst. Artif. Intell. (ISAI)*, Jun. 2016, pp. 528–532.

[14] 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," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[15] T. Hoque, J. Cruz, P. Chakraborty, and S. Bhunia, "Hardware IP trust validation: Learn (the untrustworthy), and verify," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[16] H. Salmani, "COTD: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb. 2017.

[17] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, pp. 1–6.

[18] L. Piccolboni, A. Menon, and G. Pravadelli, "Efficient control-flow subgraph matching for detecting hardware Trojans in RTL models," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–19, Oct. 2017.

[19] F. Demrozi, R. Zucchelli, and G. Pravadelli, "Exploiting sub-graph isomorphism and probabilistic neural networks for the detection of hardware Trojans at RTL," in *Proc. IEEE Int. High Level Design Validation Test Workshop (HLDVT)*, Oct. 2017, pp. 67–73.

[20] T. Han, Y. Wang, and P. Liu, "Hardware Trojans detection at register transfer level based on machine learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[21] S. A. Islam, L. K. Sah, and S. Katkoori, "A framework for hardware Trojan vulnerability estimation and localization in RTL designs," *J. Hardw. Syst. Secur.*, vol. 4, no. 3, pp. 246–262, Sep. 2020.

[22] H. S. Choo, C. Y. Ooi, M. Inoue, N. Ismail, M. Moghbel, and C. H. Kok, "Register-transfer-level features for machine-learning-based hardware trojan detection," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E103.A, no. 2, pp. 502–509, 2020.

[23] F. Wijitrisnanto, S. Sutikno, and S. D. Putra, "Efficient machine learning model for hardware Trojan detection on register transfer level," in *Proc. 4th Int. Conf. Signal Process. Inf. Secur. (ICSPIS)*, Nov. 2021, pp. 37–40.

[24] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2013, pp. 190–195.

[25] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware Trojans and maliciously affected circuits," *J. Hardw. Syst. Secur.*, vol. 1, no. 1, pp. 85–102, Mar. 2017.

[26] M. E. Aminanto, T. Ban, R. Isawa, T. Takahashi, and D. Inoue, "Threat alert prioritization using isolation forest and stacked auto encoder with day-forward-chaining analysis," *IEEE Access*, vol. 8, pp. 217977–217986, 2020.

SARWONO SUTIKNO (Member, IEEE) received the bachelor's degree in electronics from the Bandung Institute of Technology, Bandung, Indonesia, in 1984, and the M.E. and Dr.Eng. degrees in integrated systems from the Tokyo Institute of Technology, Tokyo, Japan, in 1990 and 1994, respectively. His security engineering focus includes information security management systems. He holds several professional certifications, including Indonesia Internal Auditor Professional (IIAP) from IIA, Certified in Cybersecurity (CC) from (ISC)^2, ISMS Provisional Auditor Certificate, CISA, CISSP, CISM, and CSX-F. He is also appointed as an ISACA Academic Advocate. His research interests include implementing cryptographic algorithms in integrated circuits and hardware security, including embedded system security.

SEPTAFIANSYAH DWI PUTRA received the bachelor's (S.T.) degree in electrical engineering from Lampung University and the master's degree (cum laude) in computer engineering and the Ph.D. degree (cum laude) in electrical engineering and informatics from Institut Teknologi Bandung (ITB). He is currently a Lecturer and a Researcher with the Cybersecurity Research Group, Internet Engineering Technology Program, Lampung State Polytechnic, Lampung, Indonesia. He also holds several professional certificates in cybersecurity and computer system security. His research interests include cognitive artificial intelligence and cybersecurity.

FAJAR WIJITRISNANTO received the B.S. degree (cum laude) in cryptographic engineering from National Cyber and Crypto Polytechnic, Bogor, Indonesia, in 2018, and the M.S. degree (cum laude) in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2022. From 2018 to 2019, he was a Penetration Tester working as the first deputy of the National Cyber and Crypto Agency (BSSN). From 2019 to 2020, he was a member of the Red Team in the Security Operation Center (SOC), BSSN. He is currently with BSSN, a nation-grade IT Security Consulting Institute founded by the Indonesian Government, in 2017. His work is centered on network and application security, machine learning, and hardware security.

MUHAMAD ERZA AMINANTO (Member, IEEE) received the bachelor's and master's degrees in electrical engineering from the Bandung Institute of Technology (ITB), Indonesia, in 2013 and 2014, respectively, and the Ph.D. degree from the School of Computing, Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2018. He was a Researcher in AI x Security with the National Institute of Information and Communications Technology (NICT), Tokyo, Japan, and a Lecturer in cybercrime with the University of Indonesia (UI). He is currently an Assistant Professor with the Cyber Security Program, Monash University Indonesia. He is also a Senior Research (Data) Scientist with Jakarta Smart City and an advisory board member of several government and private organizations. His current research interests include information security, artificial intelligence, anomaly detection, intrusion detection, cybersecurity, digital transformation, and smart city.