ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
Received: 5 September 2025; Accepted: 8 December 2025

https://doi.org/10.17559/TV-20250905002947
Original scientific paper

Imbalanced Hardware Trojan Detection Based on Conditional Generative Adversarial Networks

Xiang WANG, Yan LI, Xiaobo HU, Jing WANG, Yinzi TU, Meng LIU, Lixiang SHEN*

Abstract: Hardware Trojan (HT) can compromise the security of a system by changing the integrated circuit (IC) functionality and reducing the system's reliability. To handle this issue, machine learning has been widely used to analyze the datasets extracted from circuits to detect hardware Trojans. However, the extant HT detection methods provide low performance and are not applied to evaluate comprehensively using imbalanced data, which may degrade the performance of machine learning. To overcome this limitation, we proposed a conditional generative adversarial networks method that integrates the machine learning with the deep learning to detect the hardware Trojans injected in Register-Transfer Level code. A framework including feature extraction and data augmentation is proposed. Firstly, the control flow graph and data dependence graph are constructed from Register-Transfer Level code. Then, the 16 features are extracted by walking the graphs. Because there is class imbalance, a Conditional Generative Adversarial Network is proposed. Again, based on the Conditional Generative Adversarial Network model, the synthetic data is generated to balance the feature datasets. Furthermore, machine learning algorithms analyze the balanced feature datasets. The experiments use the Trust-hub benchmarks and Hummingbird e203 designs to assess our method. Finally, compared to the original datasets, the datasets enhanced by our proposed CGAN improved the F1 score and GMean of the machine learning algorithms by 32.31% and 24.17%, respectively. Moreover, when compared to the SMOTE-enhanced datasets, our method yielded a 30.51% increase in F1 score and a 21.98% increase in GMean. This demonstrates the consistency and effectiveness of our newly proposed model in detecting different types of HTs across imbalanced dataset, and it contributes to enhancing the security and trustworthiness of ICs against hardware Trojan attacks.

Keywords: conditional generative adversarial networks; data augmentation; hardware Trojan detection; imbalanced data

1 INTRODUCTION

With the rapid development of integrated circuit (IC) industry, Hardware Trojans have become one of the main threats to the security of integrated circuits. Agrawalet al [1] firstly formally analyzed the issue of hardware Trojans in Integrated Circuit supply chain. The author noted the potential for hardware Trojans injected into RSA implementations to induce errors within the Chinese Remainder Theorem (CRT) inversion step.

The subsequent two decades have witnessed an increasing of hardware Trojans detected across a diverse range of integrated circuit. This hardware security issue poses a significant threat to the security of IC. For instance, hardware Trojans injected into Wireless Network-on-Chip (WiNoC) architectures can compromise communication across shared wireless mediums, leading to security vulnerabilities such as denial of service attacks, spoofing, and eaves dropping [2]. Hardware Trojans can be embedded in AI chips that use resistive random access memory (RRAM) to alter the output parameters of hidden layers. For instance, He et al. designed hardware Trojans to dynamically affect the operation of neurons, resulting in calculation errors [3]. So hardware Trojans are an urgent issue that needs to be solved.

Hardware Trojans can be detected during two main stages: the pre-silicon phase and the post-silicon phase. The post-silicon phase, which involves testing the physical chip after manufacturing, primarily utilizes techniques such as side-channel analysis, reverse engineering, and functional testing. However, the costs associated with these post-silicon detection techniques are high. Moreover, even when hardware Trojans are detected, they cannot be removed from the manufactured chip. Compared with post-silicon detection techniques, identifying hardware Trojans during the pre-silicon phase is more cost-effective, allowing the detected malicious circuits to be fixed before fabrication.

The current literature on hardware Trojan detection in the pre-silicon phase is broadly divided into three categories: formal verification, information-flow technique, and machine learning (ML). Formal verification suffers from state-space explosion and consumes enormous computational resources, making it difficult to analyze large-scale or complex circuits. The information-flow technique is suitable for detecting information-leakage-based hardware Trojans, but it lacks sufficient capability for detecting other types of them. Compared to the other two technologies, machine learning is easier to automate; it can detect and discover new types of these Trojans and is particularly suitable for large-scale circuits with complex structures.

Class imbalance is a common issue [4, 5]. Detecting hardware Trojans using machine learning often encounters significant class imbalance, where normal data substantially outweighs abnormal data, which can severely degrade detection performance. Therefore, effectively addressing this class imbalance is important for robust hardware Trojan detection. While existing literature extensively explores hardware Trojan detection based on machine learning and deep learning, most studies concentrate on gate-level netlists [6-8] and side-channel ayalysis [9, 10], with limited focus on Register-Transfer Level (RTL) class imbalance. This is a significant oversight, as hardware engineers typically design at RTL rather than gate-level netlists. Consequently, analyses based on gate-level netlist datasets are often used to indirectly interpret the presence of hardware Trojans embedded in the original RTL designs. To bridge this gap and enhance the detection efficiency of machine learning, our work specifically addresses the class imbalance issue in the datasets extracted from RTL-level circuit designs. We propose a novel framework for detecting RTL-level hardware Trojans: First, the framework extracts feature sets from RTL-level circuit designs (such as Verilog code); then, it utilizes our proposed Conditional Generative Adversarial Network (CGAN) algorithm to augment these

datasets; and finally, it applies machine learning algorithms to detect RTL-level hardware Trojans based on the augmented datasets.

The contributions of this article are as follows.

1) A feature extracting method is proposed. The method extracts 16 features from RTL code. Then the RTL features are used to construct feature vectors.

2) In order to weaken the impact of the imbalanced RTL feature vectors, a CGAN algorithm is proposed to improve the efficiency of machine learning algorithms.

3) We evaluate the proposed CGAN using four machine learning algorithms (Random Forest, XGBoost, Bayesian Network, and Multilayer Perceptron) and analyze the experimental results.

2 LITERATURE REVIEW

2.1 Feature Extraction

Feature extraction is the first step in machine learning and has a significant impact on analysis results [i, ii,]. Wu et al. [11] mapped the hardware design onto lookup tables (LUTs) within the FPGA. From the FPGA netlists, a four-dimensional feature vector was extracted for each signal, covering structural and behavioral aspects. Faezi et al. [12] programmed circuits containing hardware Trojans onto FPGA, then captured electromagnetic and power side-channel signals. Subsequently, HTnet was designed to extract latent features from the two signals. TD-Zero [13] converted gate-level netlists into directed graphs, where nodes represented signals and logic gates represented edges. It employed self-supervised learning and metric learning, utilizing GCNs for hardware Trojan detection. Similar to TD-Zero, Pan et al. [14] mapped gate-level netlists to directed graphs. Traditional test metrics, such as combinational controllability (CC), combinational observability (CO), and static transition probability, were extracted from these graphs. To address the limitations of controllability metrics, Hu et al. [15] introduced a concealment metric that quantifies the difficulty of setting an output of a gate-level circuit to logical 0 or 1.

Compared with gate-level netlists, RTL code proposes a clearer logical idea or semantics which implements some special functions. However, few studies have discussed the feature extraction from RTL design. Yang et al. [16] extracted predominantly syntactic RTL features, such as operators (e.g., XOR) and conditional statements (if and case). Ma et al. [17] extracted seven features for hardware Trojan detection. These comprised six structural features, which were extracted from RTL code, and an additional node-type feature that was specifically defined. Fan et al. [18] proposed a method that first converted RTL code into a signal-transfer graph, and then extracted circuit-structural and graph-centrality features from the graph. The circuit-structural features primarily include inherent features, fan-in/fan-out features, constant features, and position features, while the graph-centrality features encompass degree centrality, betweenness centrality, and closeness centrality.

Therefore, our work extractS features from RTL designs by walking the control flow graph and data dependence graph.

2.2 Data Augmentation for Imbalance Data

To date, many machine learning algorithms have been used to analyze the hardware Trojan feature datasets. He et al. [3] proposed HTcather, which combined finite state machines, feature verification, and memory optimization to detect hardware security threats in neuromorphic computing systems. Sharmaet al. [19] detected hardware Trojans from IC layout images based on Deep Convolutional Neural Network. Although Faezi et al. [20] detected RS232 and AES benchmarks from Trust-Hub, its essence is to collect power consumption data and then analyze it using the Hierarchical Temporal Memory (HTM) method, achieving an average of 92.2% detection results. ResNeXt network and an attention mechanism were combined to detect side-channel hardware Trojans in AES benchmarks from Trust-Hub [21], reaching an average of 97% accuracy. A scheme including a Bayesian model and random forest algorithm was proposed to detect four types of hardware Trojans in Ref [22]. The feature data was obtained from the traffic of Gem5 Garnet 2.0, which simulated the on-chip networks with embedded hardware Trojans.

However, the positive data labelled as Trojans usually has a small number in total number of feature data because a hardware Trojan is a small circuit in RTL code or gate-level netlists. As a result, the imbalance between positive data and negative data decreases the performance of ML. This issue makes the positive class difficult to learn, often leading to misclassification. Therefore, the accuracy of ML is usually very high, but the precision and recall are very low. To solve the imbalance problem, some methods are proposed, such as oversampling, undersampling and GAN. Undersampling is unsuitable for addressing class imbalance in smallsample datasets. Since many datasets extracted from RTL code, particularly those related to hardware Trojans, have small sample sizes undersampling is therefore not suitable for addressing class imbalance in RTL-level hardware Trojans.

Oversampling techniques augment the number of minority class. SMOTE is a widely used oversampling algorithm. The augmentation process of SMOTE includes three steps. The first step is the choice of minority sample x_i. The k-nearest neighbors of x_i can be obtained from minority class. In the second step, a neighbour x_{ij} is randomly chosen from the k-nearest neighbors, and the process is repeated N times. Then a synthetic sample is obtained by interpolating between x_i and x_{ij}. Finally, N synthetic samples are generated for minority class. Based on SMOTE, some improved algorithms are proposed to address the limitations of SMOTE. T. Lavanya et al. used Vivado to analyse RTL designs (Trust-Hub AES-256) to extract some features, such as delay, power consumption and resource utilization [23]. Subsequently three SMOTE algorithms resampled the features to balance the feature data. The circuit design categories lacked diversity because only AES-256 RTL designs were analyzed. Although the method analyzed RTL code, the extracted features were primarily physical characteristics at the circuit level, failing to capture the structural and semantic features inherent in RTL designs. Liu et al. [24] augmented data using SMOTE with k-nearest neighbors, and adopted a random forest classifier. The experimental dataset was

sourced from Trust-Hub gate-level netlists, with results exhibiting significant variation in true positive rate.

In 2014, Goodfellow et al. [25] proposed Generative Adversarial Network (GAN), in which a discriminator model and a generator model were trained simultaneously, and synthetic samples were generated according to the generative model. Generative Adversarial Network (GAN) is widely applied and effective in fields requiring data generation, manipulation, and content creation. Its primary application lies in image processing, particularly for image enhancement [26-28]. Building upon the foundational GAN framework, subsequent research has explored methods to enhance its capabilities and stability. For instance, Mirza et al. [29] introduced Conditional Generative Adversarial Networks (CGANs) method, which enabled the generation of samples with specific class labels. Xu et al. [30] utilized a CGAN, mode-specific normalization, and training-by-sampling to address multimodality and class imbalance in tabular data. The experimental datasets comprised seven manually constructed datasets with specific statistical characteristics, six datasets from the UCI Machine Learning Repository, and two other commonly used public machine learning datasets.

The hardware Trojans were designed to capture data passing through the NoC switches and leaked this information to external attackers [31]. These attackers then utilized Artificial Neural Networks (ANNs) to classify the leaked data, aiming to determine the combination of running applications. To enhance the accuracy of this attack, Generative Adversarial Networks (GANs) were employed for data augmentation. This augmentation significantly improved the robustness of the attacker model. Specifically, the GAN's role in this method was to augment the data captured by the hardware Trojans.

Vishwakarma et al [32] proposed a method that first converted RTL Verilog code into graph representations and tabular data, then extracted features from the Abstract Syntax Tree. Subsequently, GANs were employed to generate the missing modalities and augmented both data categories (Trojan-Free and Trojan-Infected). Finally, the augmented samples were classified by a Convolutional Neural Network (CNN), which increased the accuracy of multimodal fusion. Nonetheless, the features remained somewhat implicit after multimodal learning, making it challenging to interpret the original Verilog design. Chen et al. [33] proposed converting gate-level netlists into undirected graphs, which extracted five features: logic gate type, port information, node degree, node betweenness centrality, and node eigenvector centrality.

In addition to SMOTE and GAN, other methods are utilized to augment data for hardware Trojan detection. Duo et al. proposed a novel approach utilizing a Variational Autoencoder (VAE) for enhanced detection of hardware Trojans embedded in RTL code implementing a space-ground integrated network (SGIN) [34]. To achieve this, recurrent neural networks (RNNs) and VAE were combined to augment the hardware Trojan datasets. The experimental validation used AES and PIC RTL designs from Trust-Hub, incorporating hardware Trojan trigger mechanisms to adapt to SGIN. Sankar et al. [35] proposed a method for data augmentation utilizing histograms. This approach processed both 29 structural features extracted from gate-level netlists and synthesis features generated by Synopsys Design Compiler (DC). Within this method, histograms were employed to facilitate the interpolation of features. Devi et al. [36] employed Adaptive Synthetic Sampling (ADASYN), an oversampling method that generated minority samples, to address class imbalance in gate-level netlists. Subsequently, instance hardness was applied for undersampling to eliminate redundant and non-informative synthetic samples.

Overall, only a limited number of papers have discussed the issue of data augmentation and feature extraction in the detection of RTL-level hardware Trojans. Modern chip designs are growing increasingly large and complex. Machine learning and deep learning are well-suited to handle this scaling increase in design complexity. Simultaneously, machine learning can be readily integrated with artificial intelligence technologies, offering powerful potential for in-depth analysis of hardware security. Therefore, to improve the efficiency of machine learning in detecting hardware Trojans, we propose a CGAN to generate synthetic RTL feature data to improve the performance of machine learning.

3 PROPOSED CGAN-BASED FRAMEWORK

3.1 Threat Model and Overall Pipeline

At the RTL design stage, an attacker can insert a few lines of code to alter the circuit's regular functions, cause denial of service, impair design reliability, or leak sensitive information. Because they are knowledgeable with the original circuit designs, adversaries may carefully construct the hardware Trojan's triggers and payloads. Such hardware Trojans are stealthy and difficult to be detected in conventional functional tests. Many factors can trigger a hardware Trojan, such as the input of specific values or sequences, clock counts, or some specific circuit states.

The work in this paper targets hard-to-trigger hardware Trojans in RTL designs. By extracting and analyzing features from RTL designs, this study detects potential hardware security vulnerabilities.

Fig. 1 shows the proposed CGAN-based framework for hardware Trojan detection in our work.

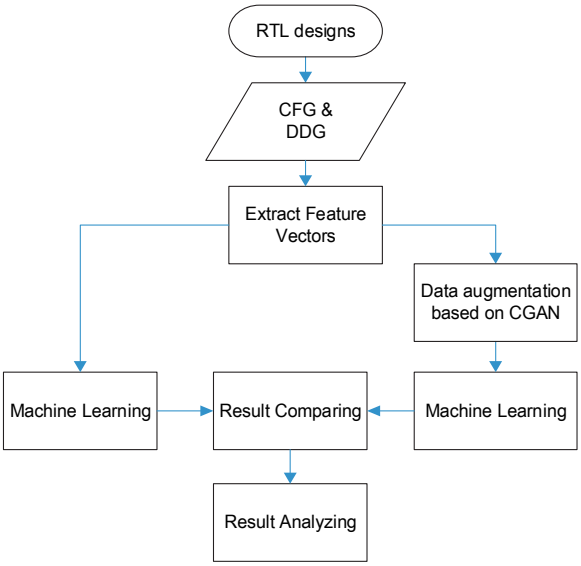

Figure 1 The CGAN-based framework for hardware Trojan detection

As shown in Fig. 1, our data augmentation framework for RTL hardware Trojan detection has three steps:

1) Extract feature vectors from RTL designs embedded hardware Trojans. 16 features are extracted from hardware design modules and labeled (0 represents a normal vector and 1 represents an abnormal one). As a result, hardware design datasets are generated.

2) A CGAN-based algorithm balances the labels on RTL design datasets.

3) Machine learning algorithms analyze the balanced datasets.

3.2 RTL Feature Extraction

RTL designs can clearly indicate various design ideas, so features can be extracted from different perspectives. To obtain RTL code features, Control Flow Graph (CFG) and Data Dependence Graph (DDG) are generated from RTL code with the method proposed in [37]. As shown in Fig. 2 and Fig. 3, RTL features can be extracted by depth-first traversal CFGs and DDGs. A CFG is generated from a Verilog module, and a DDG is generated from a whole Verilog project. A node in a CFG includes information about a statement, such as the module name, the line number in a Verilog file, the predecessor-node set, the successor-node set, and so on. And a node in a DDG includes the variables that affect the node or are affected by the node. In Fig. 2 the node name "ALWAYS-103" means that the current node represents an always statement in line 103. A node in a DDG includes the left variables of an assignment statement. The node named as "MODULE1_M_Var1" indicates a variable named "Var1" that is defined in a module named "MODULE1". The dependence relationships of variables are connected with directed lines in a DDG.

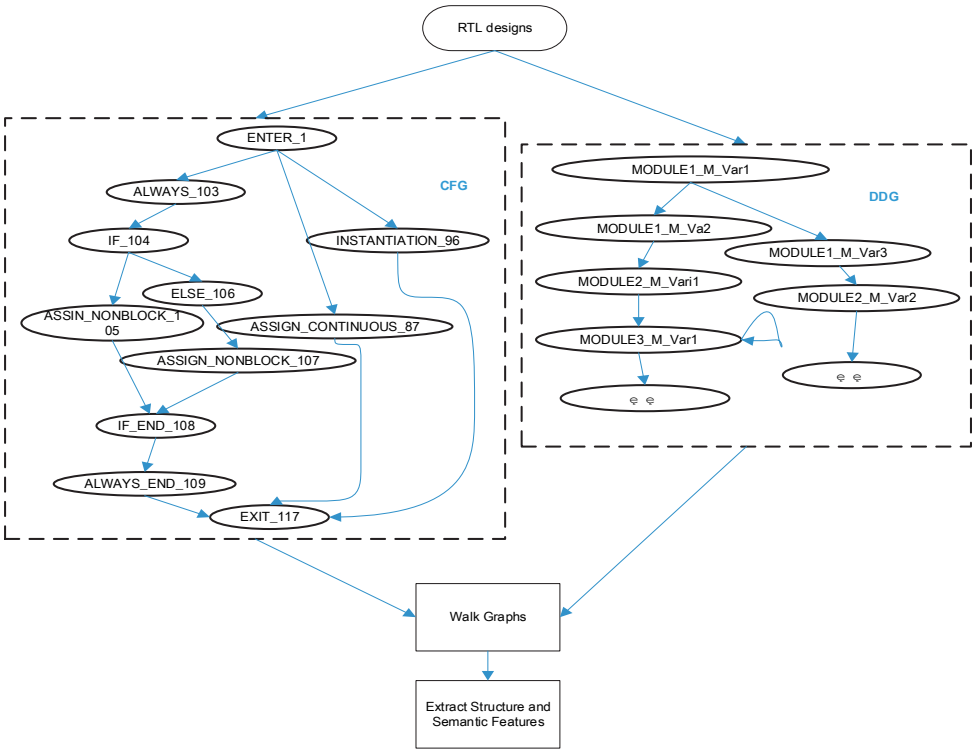

Figure 2 Extracting features from RTL designs

In Fig. 3, dashed lines and dotted lines show the process of depth-first traversal. Hardware Trojans usually have a small triggered probability, and usually affect a small number of signalling variables. These characteristics are reflected in CFGs that exhibit low node execution probability, small out-degrees and in-degrees, relations to a small number of input and output variables, etc. Therefore, 8 features are extracted by traversing the CFG (For detailed descriptions, please refer to Appendix A.1).

DDG expresses the relationship between the left variable and the right variable in an assignment expression. Hardware Trojans usually minimize their relationship with other variables in order to better hide themselves as well as accurately control activation conditions. Reflecting this in the DDG, variables in the hardware Trojan will simplify their assignment relationships with other variables. So, other 8 features are extracted by traversing CFG and DDG. (For detailed descriptions, please refer to Appendix A.2).

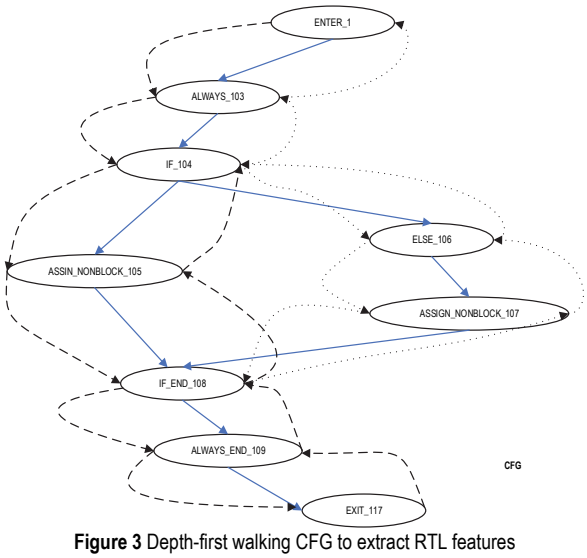

Figure 3 Depth-first walking CFG to extract RTL features

In addition, each feature vector contains the module name and line number. In this way, vectors that are detected as "positive" can be mapped to their specific location in the original Verilog file, allowing for quick localization of potential security vulnerabilities.

3.3 CGAN Model Design

The feature vectors extracted from RTL designs with embedded hardware Trojans constitute an imbalanced dataset. This is because hardware Trojans are typically implemented with a small number of statements, resulting in only a few positive labels within the dataset. Imbalanced data can negatively impact the performance and generalizability of machine learning models, while our proposed CGAN-based algorithm addresses the issue of imbalanced data by synthesizing new samples to balance feature datasets.

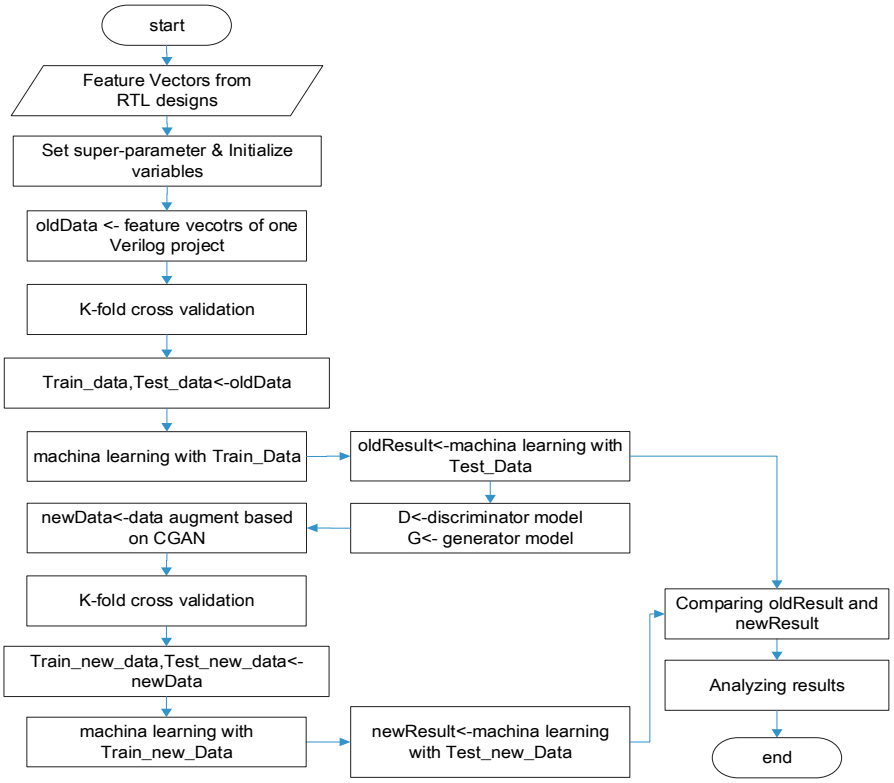

Figure 4 The CGAN-based data augmentation algorithm for hardware Trojan detection

As shown in Fig. 4, the feature vectors extracted from the RTL designs were saved as CSV files. Following the specification of hyperparameters and the initialization of relevant variables, the datasets were imported from these CSV files into *oldData*. Subsequently, a k-fold cross-validation technique was employed to partition *oldData* into training and testing sets for model evaluation. Subsequently, machine learning algorithms are employed to detect hardware Trojans within the positive dataset. These algorithms analyze *oldData* and generate a corresponding set of results *oldResult*. We then define a discriminator model and a generator model, which are integral components of our CGAN. The CGAN is leveraged to augment the positive data, thereby achieving a more balanced dataset where the number of negative labels approximates the number of positive labels. While the CGAN-based data augmentation generates *newData*. Specifically, *newData* is subjected to the same k-fold cross-validation procedure and analyzed using the identical machine learning algorithms. This process yields new result. Subsequently, a comparative analysis is conducted between old result and new result to rigorously evaluate the efficacy of our proposed CGAN-based data augmentation algorithm.

K-fold cross-validation was used in conjunction with stratified cross-validation. Stratified cross-validation is particularly well-suited for datasets exhibiting class imbalance. This technique ensures that each fold maintains a class distribution that mirrors the original dataset. This approach mitigates the risk of significant class distribution skewness within individual folds that can arise from random partitioning, thereby yielding a more robust and reliable assessment of model performance.

3.3.1 Designing the Discriminator of CGAN

The discriminator in a GAN is trained to distinguish between real data samples and synthetic data generated by the generator. Tab. 1 details the parameter configuration of the discriminator architecture. Specifically, *batch* refers to the batch size used during training. *Input_dim* and *Output dim* represent the dimensionality of the input and output layers, respectively. The dimensionality of the feature vectors within the discriminator is 16, and the parameter *num* is set to 48 (16 * 3).

The discriminator network architecture comprises a linear layer, self-attention mechanism, LeakyReLU, dropout regularization, and a final sigmoid. The linear layer performs a learned transformation of the input

features, parameterized by a weight matrix and a bias vector, with the number of input features, the number of output features, and the inclusion of a bias term defining its configuration. The self-attention mechanism aims to model long-range dependencies within the feature data. The LeakyReLU activation function introduces a small, non-zero slope for negative input values. Dropout regularization mitigates overfitting by randomly setting a proportion of neuron activation to zero during training, where this probability represents a key hyperparameter. Finally, the sigmoid activation function maps the network's output to a probability score to classify.

Table 1 The parameters of the discriminator

	Network	Parameters
Layer 1	Input	Input_dim = batch × 16,
Layer 2	Linear	Input_dim = batch × 32, Output_dim= batch × (num*16)
Layer 3	unsqueeze	Input_dim = batch × (num*16), Output_dim= batch × 1 × (num*16)
Layer 4	Self-attention	Input_dim = batch × 1 × (num*16), Output_dim= batch × 1 × (num*16)
Layer 5	squeeze	Input_dim = batch × 1 × (num*16), Output_dim= batch × (num*16)
Layer 6	Linear	Input_dim= batch × (num*16), Output_dim = batch × (num*8)
	nn.Dropout	Input_dim= batch × (num*8), Output_dim = batch × (num*8), Probability = 0.5
Layer 7	LeakyReLU	Input_dim= batch × (num*8), Output_dim = batch × (num*8)
Layer 8	Linear	Input_dim= batch × (num*8), Output_dim = batch*1
Layer 9	Sigmoid	Input_dim= batch*1, Output_dim = batch*1
	Output	batch*1

The choice of loss function was limited to the WGAN-GP [38] and Mean Squared Error (MSE), precluding a comparison with the broader spectrum of available loss functions.

For WGAN-gp, the loss of discriminator D is:

$$d\_loss = -mean(real\_out) + mean(fake\_out) + gradient\_penalty \quad (1)$$

and the loss of generator G is :

$$g\_loss = -mean(fake\_out) \quad (2)$$

where *real_out* is the output of $D(real\_x)$, *fake_out* is the output of $D(G(fake\_x))$. Gradient_penalty is gradient penalty which constrains the gradient of D not to exceed k:

$$gradient\_penalty = \lambda * MSE(GRAD(d(X\_sample), X\_sample) - k) \quad (3)$$

where λ is the parameter of gradient penalty, and *X_sample* is x sample in the space of joint distribution.

For MSE, the loss of discriminator D is:

$$d\_loss = (MSE(real\_out, real\_labels) + MSE(fake\_out, fake\_labels))/2 \quad (4)$$

and the loss of generator G is:

$$g\_loss = MSE(fake\_out, fake\_labels) \quad (5)$$

3.3.2 Designing the Generator of CGAN

The parameters of the CGAN generator list in Tab. 2. The generator network leverages a combination of linear layers, dropout regularization, and leaky ReLU. BatchNorm1d (Batch Normalization) is incorporated to rescale the inputs, promoting stable generalization performance. It is important to note that BatchNorm1d involves several configurable parameters, including the number of features and a small constant (epsilon) for numerical stability. *Z_dim* denotes the dimensionality of the random noise vector used as input.

Table 2 The parameters of the generator

	Network	Parameters
Layer 1	Input	batch × (z_dim)
Layer 2	Linear	Input_dim = batch × (z_dim+ z_dim), Output_dim= batch × (num*4)
Layer 3	LeakyReLU	Input_dim= batch × (num*4), Output_dim = batch × (num*4)
Layer 4	Linear	Input_dim= batch × (num*4), Output_dim = batch × (num*8)
Layer 5	nn.BatchNorm1d	Input_dim= batch × (num*8), Output_dim = batch × (num*8)
Layer 6	LeakyReLU	Input_dim= batch × (num*8), Output_dim = batch × (num*8)
Layer 7	Linear	Input_dim= batch × (num*8), Output_dim = batch × (num*16)
Layer8	nn.BatchNorm1d	Input_dim= batch × (num*16), Output_dim = batch × (num*16)
Layer 9	LeakyReLU	Input_dim= batch × (num*16), Output_dim = batch × (num*16)
Layer 10	Linear	Input_dim= batch × (num*16), Output_dim =batch × (num*16)
Layer 11	nn.BatchNorm1d	Input_dim= batch × (num*16), Output_dim = batch × (num*16)
Layer 12	LeakyReLU	Input_dim= batch × (num*16), Output_dim =batch × (num*16)
Layer 13	Linear	Input_dim= batch × (num*16), Output_dim =batch × (num*8)
Layer 14	nn.BatchNorm1d	Input_dim= batch × (num*8), Output_dim = batch × (num*8)
Layer 15	LeakyReLU	Input_dim= batch × (num*8), Output_dim =batch × (num*8)
Layer 16	Linear	Input_dim= batch × (num*8), Output_dim = batch*16
	Output	batch*16

3.4 The Training/Validation Process

Algorithm 1 details the complete data augmentation process. Following the specification of hyperparameters, the selection of a machine learning algorithm, and the definition of a loss function, the data from file *f* is loaded into the variable *oldData*. *F* is a dataset extracted from a RTL code file. Subsequently, the test function analyzes the training and testing datasets extracted from *oldData*. The returned results *oldResult* includes evaluation metrics, such as accuracy, precision, recall, specificity, F1-score, and GMeans. The CGAN proposed in this study utilizes a generator *G* that is trained via the *train_CGAN* function. Subsequently, leveraging this trained generator *G*, the *test WithGen* function synthesizes artificial data. The synthetic data is then subjected to a machine learning algorithm. The resulting evaluation metrics *newResult* mirror those obtained from *oldData*, facilitating a direct comparison.

Finally, these results are graphically presented to assess the efficacy of the CGAN.

Algorithm 1: The whole algorithm	
Input: feature datasets extracted from RTL code	
Output: classifying metrics	
1	set hyper-parameters
2	select a machine learning algorithm
3	set loss function
4	for each *f* in dataset_file_list:
5	*oldData* <- read data from *f*
6	#analyze *oldData* using ML *oldResult* <- test(*oldData*)
7	#train generator *G* using our CGAN generator *G* <- train_CGAN(*oldData*)
8	#analyze *newData* using ML *newResult* <- test *WithGen*(*oldData*, *G*)
9	plot all *oldResult* and *newResult*

Algorithm 2 details the *test* function, which orchestrates the analysis of data using a suite of classifiers. Specifically, the *randomForest* function analyzes the input data *dataset*, generates predictions, and subsequently outputs a range of evaluation metrics *result*. The other functions perform analogous data processing and evaluation procedures.

Algorithm 2: test() # classify data with machine learning	
Input: feature dataset	
Output: classifying metrics	
1	if algorithm == 'randomforest':
2	result <- randomforest(dataset)
4	if algorithm == 'xgboost':
5	result <- xgboost(dataset)
6	if algorithm == 'bayesianNetwork:
7	result <- bayesianNetwork(dataset)
8	if algorithm == 'mlp': # Multilayer Perceptron
9	result <- mlp(dataset)
10	return result

Algorithm 3 details the training procedure for the discriminator *D* and generator *G* models, utilizing one-hot encoded labels for the input datasets. Specifically, lines 3-5 outline the preparation of the training datasets. Subsequently, the discriminator *D* and the generator *G* are trained.

Algorithm 4 leverages the trained generator *G* to produce synthetic data *fakeData*. Subsequently, the generated data is subjected to further analysis through the test function. Finally, the classifying metric *newResult* is obtained.

Algorithm 3: train_CGAN() # train the models of CGAN	
Input: a data set, discriminator *D*, generator *G*	
Output: *d_loss*, *g_loss*, generator *G*	
1	for number of training iterations:
2	for number of training batch:
3	*x_lab* <- get labels of dataset
4	*x_data* <- get data of dataset
5	# transform labels (0, 1) to one hot encoding *y_one_hot* <- onehot(*x_lab*)
6	train discriminator *D* with *y_one_hot*
7	train generator *G*
8	return *d_loss*, *g_loss*, generator *G*

Algorithm 4: testWithGen()	
Input: generator *G*, *oldData*	
Output: classifying metrics	
1	*fakeData* <- generator synthetic data by *G*
2	*newData* <- *oldData* + *fakeData*
3	*newResult* <- test(*newData*)
4	return *newResult*

4 EXPERIMENTAL EVALUATION AND RESULTS

4.1 Evaluation Metrics

It is noted that TP (True Positive), TN (True Negative), FP (False Positive) and FN (False Negative) are commonly employed in evaluating machine learning algorithms, the subsequent metrics discussed are derived from these fundamental measures.

1) Accuracy:

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

2) Precision:

$$\frac{TP}{TP+FP} \tag{7}$$

3) Recall or TPR (True Positive Rate):

$$\frac{TP}{TP+FN} \tag{8}$$

4) Specificity:

$$\frac{TN}{TN+FP} \tag{9}$$

5) F1-score:

$$\frac{2\times presion\times recall}{presion+recall} \tag{10}$$

6) GMean:

$$\sqrt{recall\times specificity} \tag{11}$$

4.2 The Verilog Design for Experiments

For experimental evaluation, the RS-232 and wb_conmax designs available from Trust-Hub were chosen. Furthermore, to enhance the diversity of benchmarks, several distinct modules derived from the Hummingbird e203 processor were selected and utilized to embed the hardware Trojans. The primary rationale for selecting the Hummingbird e203 design is predicated on the following considerations.

1) Hummingbird E203 represents a well-designed RISC-V processor core, encompassing the essential functional blocks typically found in embedded processors.

2) The Hummingbird e203 implements a system-on-chip (SoC) design.

3) The Hummingbird e203 has been applied in practice.

The hardware Trojans was carefully designed and embedded into the Hummingbird e203 design. The validity of the hardware Trojans was confirmed using simulation tests and NuSMV verification. Consequently, six modules were selected for the evaluation of our CGAN.

4.3 The Experiments Results Analysis

4.3.1 Imbalance Ratio

To extract relevant features, 16 RTL designs were selected. Hardware Trojan feature vectors were assigned a positive label, while feature vectors from normal designs were assigned a negative label. The specific number of positive and negative labels used in this study is detailed in Tab. 3.

Table 3 The imbalance percent of 16 RTL designs

RTL design	Positive	Negative	Imbalance ratio / %
UART-T300	18	259	6.95
UART-T800	4	261	1.53
UART-T200	13	263	4.94
UART-T600	56	259	21.62
UART-T400	20	253	7.91
UART-T700	57	261	21.84
UART-T900	64	258	24.81
UART-T500	28	255	10.98
UART-T100	6	260	2.31
Wb_conmax-T300	7	1180	0.59
Wb_conmax-T200	21	1181	1.78
e203_exu_alu	29	94	30.85
e203_exu_decode	8	469	1.71
e203_itcm_ctrl	11	47	23.4
jtag_interface	6	71	8.45
sirv_gnrl_fifo	16	30	53.33
sirv_gnrl_icb_n2w	6	30	20.0

The imbalance ratio (%) represents the ratio of the positive samples to the negative samples. During the training process, the learning rates for the discriminator and generator were configured to $1e^{-4}$ and $5e^{-4}$, respectively. Furthermore, the dimensionality of the generator's input noise vector (*Z-dim*) was set to 50, and the total number of training epochs was set to 200.

4.3.2 Evaluation Metrics Analysis

The experiment first attempted dimensionality reduction using principal component analysis (PCA). While PCA offers a potential solution to mitigate performance degradation following dimensionality reduction in some datasets, this algorithm was not adopted in this study. Instead, to preserve the nuances inherent in the data and avoid potential information loss associated with feature transformation, all 16 features were retained for subsequent machine learning analyses.

This study compared three optimizers (SGD, Adam, RMSprop), and two loss functions (gradient penalty, MSE) to identify the optimal combination. The combination of SGD optimizer and MSE loss function yielded the highest detection score. Therefore, the experiments in this paper adopted SGD and MSE.

In this study, four algorithms are used to evaluate the performance of our proposed CGAN algorithm: Random Forest, XGBoost, Bayesian Network, and Multilayer Perceptron (MLP). Tab. 4 presents the classification metrics for the four algorithms under three distinct datasets (original data, our CGAN-based augmented data, SMOTE-based augmented data).

Following data augmentation via the proposed algorithm, a consistent improvement in F1-score and G-mean was observed across all four algorithms. While specificity experienced a slight decrease in three instances, this reflects a minor decrease in the accuracy of negative instance classification post-augmentation. This outcome underscores the inherent class imbalance within the original dataset, which leads to a bias in the classification results towards the majority negative class, ultimately reducing the classification accuracy for positive instances.

Table 4 The comparison of classifying results

	metric	Original data	Our CGAN Improvement		SMOTE Improvement	
RandomForest	Accuracy	96.24	97.09	0.85	94.26	–1.98
	Recall	64.56	96.51	31.95	67.66	3.10
	Precision	76.06	97.84	21.78	78.92	2.86
	Specificity	98.88	97.53	–1.35	97.37	–1.51
	F1	66.43	97.09	30.66	69.34	2.91
	GMean	71.91	96.46	24.55	76.43	4.51
XGBoost	Accuracy	96.36	97.48	1.12	96.89	0.53
	Recall	75.65	97.13	21.48	69.52	–6.13
	Precision	89.90	98.06	8.16	95.58	5.68
	Specificity	98.98	97.94	–1.04	99.43	0.45
	F1	76.7	97.54	20.84	72.24	–4.46
	GMean	80.08	98.17	18.09	74.67	–5.41
Bayesian Network	Accuracy	83.97	95.12	11.15	84.55	0.58
	Recall	82.89	94.10	11.21	87.73	4.84
	Precision	61.53	96.79	35.26	59.19	–2.34
	Specificity	83.63	95.59	11.96	83.18	–0.45
	F1	61.52	95.32	33.80	62.93	1.41
	GMean	78.37	94.93	16.56	82.02	3.65
MLP	Accuracy	94.06	96.65	2.59	95.53	1.47
	Recall	48.24	95.59	47.35	56.76	8.52
	Precision	85.18	98.12	12.94	92.94	7.76
	Specificity	98.06	97.82	–0.24	98.12	0.06
	F1	52.76	96.71	43.94	60.12	7.36
	GMean	58.24	95.71	37.47	64.24	6.00

While the SMOTE algorithm augments the original dataset, in seven instances, the indicator values were lower compared to the original dataset. Although marginal improvements are observed across most indicator values, the overall algorithmic performance demonstrates only a limited improvement, as illustrated in Fig. 5. Specifically, Fig. 5 presents a bar chart derived from the data in Tab. 4, offering a clear comparative visualization of the average efficiency achieved by the four algorithms across the three distinct data types. The imblearn package in Python provides several oversampling algorithms, such as SMOTE, SVMSMOTE, SMOTENC, SMOTEN and ADASYN. As our experiments showed that these algorithms yielded nearly identical results, so SMOTE was selected for generating the result figures.

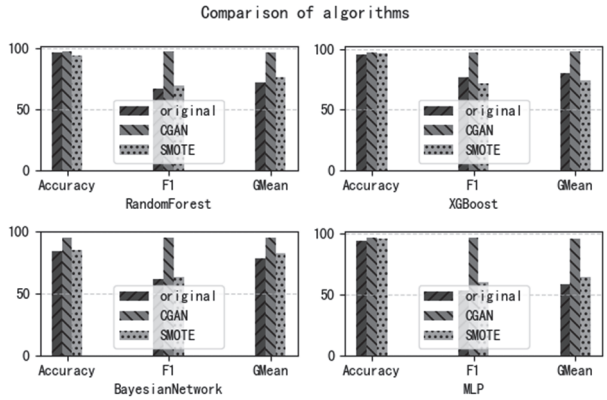

Figure 5 Comparison of algorithms

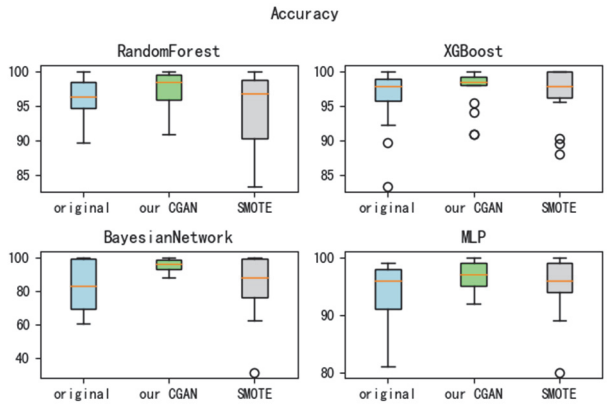

Figure 6 Accuracy for 16 RTL designs

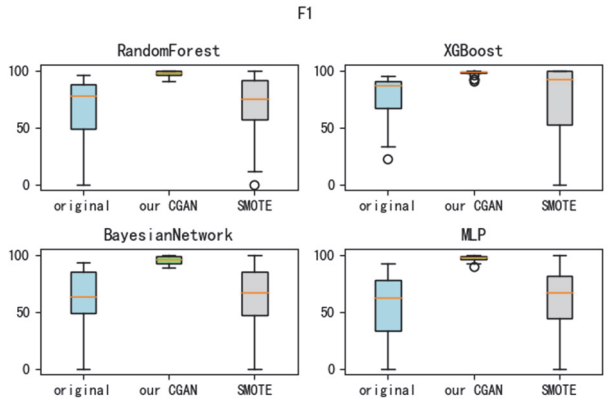

Figure 7 The F1-scores for 16 RTL designs

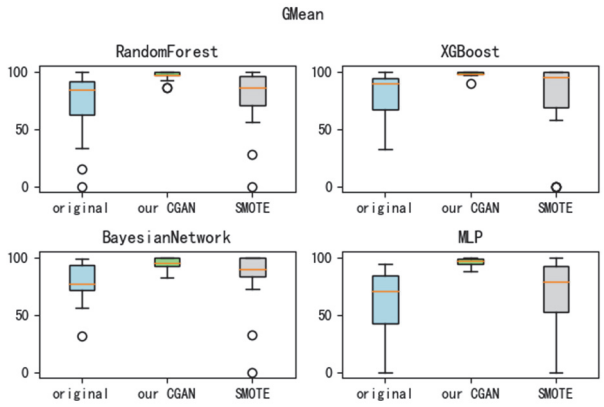

Figure 8 The GMean for 16 RTL designs

Fig. 9 and Fig. 10 present the F1-scores obtained by the four machine learning algorithms on the 16 RTL designs. The results show that the dataset augmented by our proposed CGAN algorithm yields a much smaller variation in detection performance than the datasets augmented by SMOTE.

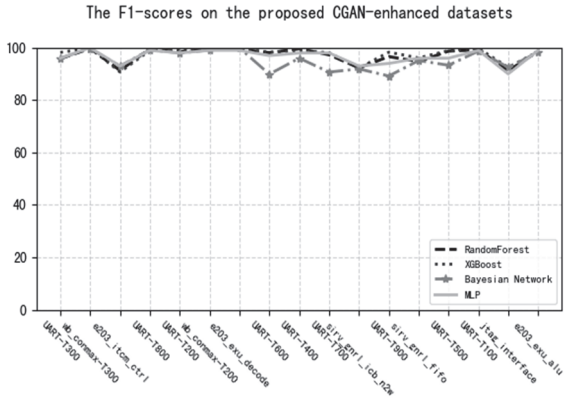

Figure 9 The F1-scores on the proposed-enhanced datasets

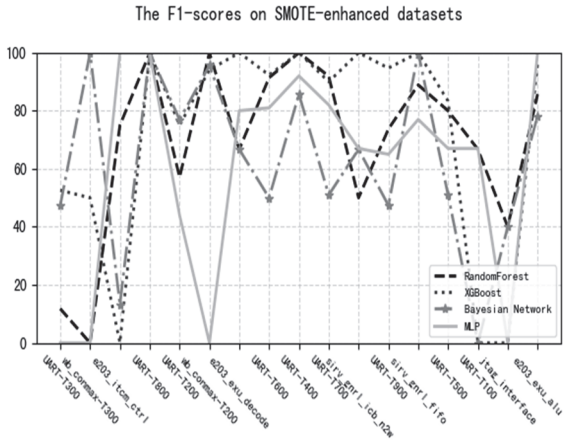

Figure 10 The F1-scores on SMOTE-enhanced datasets

The essence of CGAN data augmentation involves generating high-quality synthetic minority samples to adjust the decision boundary in the feature space, which optimizes metrics focused on minority class performance. By generating synthetic minority samples, our CGAN balances the class distribution in benchmark datasets. This enables the classifier to learn a more equitable decision boundary, which would otherwise be severely biased toward the majority class. Our proposed CGAN enhances recognition capabilities for minority classes, improving

recall for these classes. Furthermore, since the generated samples accurately reflect the intrinsic characteristics of the minority class, they prevent the decision boundary from being biased toward the majority class. This preserves high minority class precision, reducing the misclassification of majority samples as minority. The improvement in minority recall thereby improves the GMean; the simultaneous improvement of recall and precision results in an increase in the F1 score, demonstrating that our model achieves more balanced classification performance overall.

While data enhancement utilizing the proposed CGAN demonstrated improvements over SMOTE-based enhancement, the magnitude of these gains warrants careful consideration. Specifically, Tab. 5 summarizes the metric averages for the three data types, and Tab. 6 exhibits the overall average improvement of the proposed CGAN-enhanced datasets compared to the original datasets and the SMOTE-enhanced datasets.

Table 5 The average metric score

RTL design	Original data	Our CGAN	SMOTE
Accuracy	92.66	96.59	92.81
Recall	67.84	95.83	70.42
Precision	78.17	97.70	81.66
Specificity	94.89	97.22	94.53
F1	64.35	96.67	66.16
GMean	72.15	96.32	74.34

Table 6 The average improvement of the proposed CGAN

RTL design	Our CGAN - Original data	Our CGAN - SMOTE
Accuracy	3.93	3.78
Recall	28.00	25.42
Precision	19.54	16.05
Specificity	2.33	2.70
F1	32.31	30.51
GMean	24.17	21.98

5 CONCLUSIONS

This study introduces a framework to enhance feature datasets derived from hardware Trojans injected into RTL code. The proposed CGAN effectively improves the performance of machine learning algorithms in analyzing imbalanced data associated with these RTL-level Trojans. Our study shows that the proposed CGAN method can effectively improve the performance of machine learning algorithms to analyze unbalanced data, with significant improvements in both F1 and GM, yielding improvements for over 16% compared to the original datasets and more than 12% compared to SMOTE-enhanced datasets. This framework provides an effective data augmentation strategy for RTL-level hardware Trojan detection, demonstrating capability in identifying hardware Trojans across diverse circuit designs.

The proposed method primarily targets hardware Trojan detection on small sample datasets extracted from RTL-level designs. Consequently, the detection capability of the proposed CGAN for very-large-scale integrated circuits is limited.

As the scale and complexity of chip designs increase, hardware Trojan detection faces increasingly complex imbalance issues. To address these challenges, data augmentation, particularly when enhanced by AI-driven generative methods, becomes crucial for efficiently expanding training datasets for hardware Trojan detection. Therefore, further research needs to address the scope of adaptation of dataset extraction methods that can be used to characterize control information and dependencies for gate-level netlists. In combination with LLM techniques, control and dependency relationships can be constructed from the beginning of the chip's specification design. Based on this, the consistency and security detection are carried out based on these criteria from the beginning of the chip design, which provides the detection and evaluation basis for the security vulnerability detection of the design during the pre-silicon phase.

Acknowledgements

This work was supported by Laboratory Specialized Scientific Research Projects of Beijing Smart-chip Microelectronics Technology Co., Ltd. under grand number SGSC0000AQQT2401529.

6 REFERENCES

[1] Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P., & Sunar, B. (2007). Trojan detection using IC fingerprinting, *2007 IEEE Symposium on Security and Privacy, Berkeley*, 296-310. https://doi.org/10.1109/SP.2007.36

[2] Vashis, A. T., Keats, A., Pudukotai, S., & Ganguly, A., (2019). Securing a wireless network-on-chip against jamming-based denial-of-service and eavesdropping attacks. *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, *27*(12), 2781-2791. https://doi.org/10.1109/TVLSI.2019.2928960

[3] He, G., Dong, C., Huang, X., Guo, W., Liu, X., & Ho, T. (2020). HTcatcher: finite state machine and feature verifcation for large-scale neuromorphic computing systems. *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 415-420. https://doi.org/10.1145/3386263.3406955

[4] Liu, Q., Yun, F., Dong, M., Djoric, D., & Zivlak, N. (2024). Health Prognosis for Equipment Based on ACO-K-Means and MCS-SVM under Small Sample Noise Unbalanced Data. *Tehnicki vjesnik - Technical Gazette*, *31*(1), 24-31. https://doi.org/10.17559/TV-20230505000608

[5] Lai, W. (2023). Default Prediction of Internet Finance Users Based on Imbalance-XGBoost. *Tehnicki vjesnik - Technical Gazette*, *30*(3), 779-786. https://doi.org/10.17559/TV-20230302000395

[6] Salmani, H. (2022). Gradual-N-Justification (GNJ) to Reduce False-Positive Hardware Trojan Detection in Gate-Level Netlist. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *30*(4), 515-525. https://doi.org/: 10.1109/TVLSI.2022.3143349

[7] Liakos, K., Georgakilas, G., Moustakidis, S., Sklavos, N., & Plessas, F. (2020). Conventional and machine learning approaches as countermeasures against hardware trojan attacks. *Microprocessors and Microsystems*, *79*, 103295. https://doi.org/10.1016/j.micpro.2020.103295

[8] Salmani, H. (2017). COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist. *IEEE Transactions on Information Forensics and Security*, *12*(2), 338-350. https://doi.org/10.1109/TIFS.2016.2613842

[9] Lee, D., Lee, J., Jung, Y., Kauh, J., & Song, T. (2024). Robust Hardware Trojan Detection Method by Unsupervised Learning of Electromagnetic Signals. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *32*(12), 2327-2340. https://doi.org/10.1109/TVLSI.2024.3458892

[10] Bhatta, N. P. & Amsaad, F. (2025). Ml assisted techniques in power side channel analysis for Trojan classification. *Cluster Computing*, *28*, 157.

https://doi.org/10.1007/s10586-024-04715-w

[11] Wu, L., Zhang, X., Wang, S., & Hu, W. (2022). Hardware Trojan Detection at LUT: Where Structural Features Meet Behavioral Characteristics. *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 121-124. https://doi.org/10.1109/HOST54066.2022.9840276

[12] Faezi, S., Yasaei, R., & Al Faruque, M. A. (2021). HTnet: Transfer Learning for Golden Chip-Free Hardware Trojan Detection. *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1484-1489. https://doi.org/10.23919/DATE51398.2021.9474076

[13] Pan, Z. & Mishra, P. (2024). TD-Zero: Automatic Golden-free hardware trojan detection using zero-shot learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *43*(7), 1998-2011. https://doi.org/10.1109/TCAD.2024.3354889

[14] Pan, G., Li, H., & Wang, J. (2025). A fast hardware Trojan detection method with parallel clustering for large-scale gate-level netlists. *Computers & Security*, *157*, 104570. https://doi.org/10.1016/j.cose.2025.104570

[15] Hu, X., Zhang, Y., Liu, S., Chen, X., Wang, Y., Zhao, Z, Guo, Y., & Li, K. (2026). SubG4TJ: A collaborative subgraph classification method based on multidimensional attributes for hardware trojan detection. *Expert Systems with Applications*, *297*, 29355. https://doi.org/10.1016/j.eswa.2025.129355

[16] Yang, J., Zhang, Y., Hua, Y., Yao, J., Mao, Z., & Chen, X. (2021). Hardware Trojans Detection Through RTL Features Extraction and Machine Learning. *2021 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. https://doi.org/10.1109/AsianHOST53231.2021.9699658

[17] Ma, P., Shang, G., Liu, H. J., Shi, J. Y., Pan, W. T., Zhang, Y., & Hao, Y. (2025). GNN-Based Hardware Trojan Detection at Register Transfer Level Leveraging Multiple-Category Features. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *33*(3), 831-840. https://doi.org/10.1109/TVLSI.2024.3513218

[18] Fan, R., Tang, Y., Sun, H., Liu, J., & Li, H. (2024). An Efficient ML-based Hardware Trojan Localization Framework for RTL Security Analysis. *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 1-7. https://doi.org/10.1145/3670474.3685955

[19] Sharma, R., Rathor, V., Sharma, G., & Pattanaik, M. (2021). A new hardware trojan detection technique using deep convolutional neural network. *Integration*, *79*, 1-11. https://doi.org/10.1016/j.vlsi.2021.03.001

[20] Faezi, S., Yasaei, R., Barua, A., & Faruque, M. (2021). Brain-inspired golden chip free hardware trojan detection. *IEEE Transactions on Information Forensics and Security*, *16*, 2697-2708. https://doi.org/10.1109/TIFS.2021.3062989

[21] Chen, S., Wang, T., Huang, Z., & Hou, X. (2023). Detection method of golden chip-free hardware trojan based on the combination of ResNeXt structure and attention mechanism. *Computers & Security*, *134*, 103428. https://doi.org/10.1016/j.cose.2023.103428

[22] Ding, R., Zhang, Y., Xia, S., Wang, Z., Deng, J., & Chen, X. (2024). Detection of hardware attacks in network on chip based on machine learning. *2024 IEEE 7th International Conference on Electronic Information and Communication Technology (ICEICT)*, 440-445.

[23] Lavanya, T. & Rajalakshmi K. (2023). Heterogenous ensemble learning driven multi-parametric assessment model for hardware Trojan detection. *Integration*, *89*, 217-228. https://doi.org/10.1016/j.vlsi.2022.12.011

[24] Liu, Y., Li, J., Guo, P., Zhu, C., Wan, J., Zhong, J., & Zhang, J. (2024). A feature-adaptive and scalable hardware trojan detection framework for third-party IPs utilizing multilevel feature analysis and random forest. *Journal of Electronic Testing*, *40*(6). https://doi.org/10.1007/s10836-024-06150-6

[25] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems*, *27*, 2672-2680. https://doi.org/10.48550/arXiv.1406.2661

[26] Wang, X. (2024). GAN-DNADE: image encryption algorithm based on generative adversarial network and dna dynamic encoding. *Computer Science and Information Systems*, *21*(4), 1673-1697.

[27] Gowthami, S. & Harikumar, R. (2023). Residual generative adversarial adaptation network for the classification of melanoma. *International Journal of Computers Communications & Control*, 18(6). https://doi.org/10.15837/ijccc.2023.6.5274

[28] Gong, F. (2023). A Generative Adversarial Networks Based Approach for Literary Translation. *Tehnicki vjesnik - Technical Gazette*, *30*(3), 921-929. https://doi.org/10.17559/TV-20221222092039

[29] Mirza, M. & Osindero, S. (2014). Conditional generative adversarial nets. *Computer Science*, 2672-2680. https://doi.org/10.48550/arXiv.1411.1784

[30] Xu, L., Cuesta, A., Skoularidou, M., & Veeramachaneni, K. (2020). Modeling Tabular Data using Conditional GAN. *Advances in Neural Information Processing Systems 32, 10, 32nd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 8-14.

[31] Dhavlle, A., Ahmed, M., Mansoor, N., Basu, K., Ganguly, A., & Pudukotai, S. (2023). Defense against on-chip Trojans enabling traffic analysis attacks based on machine learning and data augmentation. (2023). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *42*(12), 4681-4694. https://doi.org/10.1109/TCAD.2023.3278618

[32] Vishwakarma, R. & Rezaei, A. (2024). Uncertainty-Aware Hardware Trojan Detection Using Multimodal Deep Learning. *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1-6. https://doi.org/10.23919/DATE58400.2024.10546558

[33] Chen, L. H., Dong, C., Wu, Q. W., Liu, X. M., Guo, X. D., Chen, Z. Y., Zhang, H., & Yang, Y. (2025). GNN4HT: a two-stage GNN-based approach for hardware trojan multifunctional classification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *44*(1), 172-185. https://doi.org/10.1109/TCAD.2024.3428469

[34] Zhang, D., Li, F. H., Guo, Y. C., Mao, M., & Li, Z. F. (2025). HT-ASAF: Automatic Sample Augmentation Framework for Hardware Trojan. *IEEE Internet of Things Journal*, *12*(3), 2609-2622. https://doi.org/10.1109/JIOT.2024.3474970

[35] Sankar, V., Balachander, S., Nirmala Devi, M., & Jayakumar, M. (2023). Reliability Enhancement of hardware trojan detection using histogram augmentation technique. *Proceedings of the IEEE International Conference on VLSI Design*, *2023-January*, 365-370. https://doi.org/10.1109/VLSID57277.2023.00079

[36] Nirmala Devi M. & Sankar, V. (2023). Graph based heterogeneous feature extraction for enhanced hardware Trojan detection at gate-level using optimized XGBoost algorithm. *Measurement*, *220*, 113320. https://doi.org/10.1016/j.measurement.2023.113320

[37] Shen, L. X., Mu, D., Cao, G., Qin, M., Blackstone, J., & Kastner, R. (2018). Symbolic execution based test-patterns generation algorithm for hardware Trojan detection. *Computers & Security*, *78*, 267-280. https://doi.org/10.1016/j.cose.2018.07.006

[38] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Aaron Courville, A. (2017). Improved training of Wasserstein GANs. *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*, 5769-5779.

Contact information:

Xiang WANG, MS
Beijing Smart-Chip Microelectronics Technology Co., Ltd.
Beijing City, Changping District, China
E-mail: wangxiang@sgchip.sgcc.com.cn

Yan LI
Beijing Smart-Chip Microelectronics Technology Co., Ltd.,
Beijing City, Changping District, China
E-mail: liyan@sgchip.sgcc.com.cn.

Xiaobo HU
State Grid Key Laboratory of Power Industrial Chip Design and Analysis Technology, Beijing Smart-Chip Microelectronics Technology Co., Ltd.,
Beijing City, Changping District, China
E-mail: huxiaobo@sgchip.sgcc.com.cn.

Jing WANG
State Grid Key Laboratory of Power Industrial Chip Design and Analysis Technology,
Beijing Smart-Chip Microelectronics Technology Co., Ltd.,
Beijing City, Changping District, China
E-mail: wangjing13@sgchip.sgcc.com.cn.

Yinzi TU
State Grid Key Laboratory of Power Industrial Chip Design and Analysis Technology, Beijing Smart-Chip Microelectronics Technology Co., Ltd.,
Beijing City, Changping District, China
E-mail: tuyinzi@sgchip.sgcc.com.cn.

Meng LIU
State Grid Key Laboratory of Power Industrial Chip Design and Analysis Technology, Beijing Smart-Chip Microelectronics Technology Co., Ltd.,
Beijing City, Changping District, China;
E-mail: liumeng@sgchip.sgcc.com.cn.

Lixiang SHEN, PhD, Assistant Professor
(Corresponding author)
School of Computer Science and Information Engineering,
Changzhou Institute of Technology,
No.666 Liaohe Road, Changzhou, Jiangsu Province, P. R. China
E-mail: slxiang_001@163.com, Shenlx@czu.cn

A.1 8 Features are Extracted by Traversing the CFG

1) NodeProbabilityFeature

NodeProbabilityFeature shows the brunch probabilities of a conditional statement node, such as "if...else" or "ca-se". NodeProbabilityFeature of an instantiation statement node or an assign statement node is 1.

2) ExecuteProbabilityFeature

If there is a path between ENTER and the current node, executeProbabilityFeature is the product of nodeProbabilityFeature of nodes in the path.

3) PreNodeNumFeature

The number of nodes that are predecessors of a CFG node is preNodeNumFeature. ENTER has only successors.

4) NextNodeNumFeature

The number of nodes that are successors of a CFG node is nextNodeNumFeature. EXIT has only predecessors.

5) ExecuteDepthFeature

If a path exists between ENTER and a node, the path length is the number of nodes on the path. The shortest path length is executeDepthFeature.

6) EffectedByInputNumFeature

Let the predecessors of a node be PRE_NODE. Then the number of input variables that exist in PRE_NODE is effected by InputNumFeature.

7) EffectOutputNumFeature

Let the successor nodes of a node be SUCC_NODE. Then the number of output variables that exist in SUCC_NODE is effectOutputNumFeature.

8) NodeTypeFeature

NodeTypeFeature represents the type of a Verilog statement. The types include ASSIGN_CONTINUOUS, ASSIGN_BLOCKING, ASSIGN_NONBLOCKING, IF, CASE, and so on.

A.2 8 Features are Extracted by Traversing CFG and DDG

1) LeftVariableRightNumberFeature

The leftVariableRightNumberFeature represents the number of times a variable, which appears on the left-hand side of an assignment statement in a CFG node, also appears on the right-hand side of a statement in a DDG node.

2) LeftVariableLeftNumberFeature

The leftVariableLeftNumberFeature represents the number of times a variable, which appears on the left-hand side of an assignment statement in a CFG node, also appears on the left-hand side of a statement in a DDG node.

3) VariableSetLeftNumberFeature

The variableSetLeftNumberFeature represents the number of times a variable, which appears on the right-hand side of an assignment statement in a CFG node, also appears on the left-hand side of a statement in a DDG node.

4) LeftVariableControlNumberFeature

The leftVariableLeftNumberFeature represents the number of times a variable, which appears on the left-hand side of an assignment statement in a DDG node, also appears on the CFG control nodes.

5) ControlDepthFeature

The controlDepthFeature is defined as follows: for each input variable within the DDG, we identify all paths leading to a node that corresponds to a variable on the left-hand side of an assignment within the CFG. The length of each identified path is then defined as the number of nodes traversed. The controlDepthFeature for a given left-hand side variable is ultimately calculated as the average length of all such identified paths.

6) ObserveDepthFeature

The observeDepthFeature quantifies the average path length between an assignment variable in the CFG and output variables in the DDG. Specifically, if a path exists from a variable on the left-hand side of an assignment within a CFG node to an output variable in the DDG, the path length is defined as the number of nodes traversed. The observeDepthFeature then represents the arithmetic mean of these path lengths, calculated across all such identifiable paths.

7) PreVariableNumberFeature

The preVariableNumberFeature is the count of DDG variables that affect a given left-hand side variable within a specific CFG node.

8) NextVariableNumberFeature

The nextVariableNumberFeature is the count of DDG variables affected by a variable appearing on the left-hand side of an assignment within a given CFG node.