electronics

Article

GAINESIS: Generative Artificial Intelligence NEtlists SynthesIS

Konstantinos G. Liakos [1], Georgios K. Georgakilas [1], Fotis C. Plessas [1] and Paris Kitsos [2,*]

[1] Electrical & Computer Engineering Department, University of Thessaly, 38334 Volos, Greece; kliakos@e-ce.uth.gr (K.G.L.); ggeorgakila@upnet.gr (G.K.G.); fplessas@e-ce.uth.gr (F.C.P.)
[2] Electrical & Computer Engineering Department, University of the Peloponnese, 26334 Patras, Greece
* Correspondence: kitsos@uop.gr

Abstract: A significant problem in the field of hardware security consists of hardware trojan (HT) viruses. The insertion of HTs into a circuit can be applied for each phase of the circuit chain of production. HTs degrade the infected circuit, destroy it or leak encrypted data. Nowadays, efforts are being made to address HTs through machine learning (ML) techniques, mainly for the gate-level netlist (GLN) phase, but there are some restrictions. Specifically, the number and variety of normal and infected circuits that exist through the free public libraries, such as Trust-HUB, are based on the few samples of benchmarks that have been created from circuits large in size. Thus, it is difficult, based on these data, to develop robust ML-based models against HTs. In this paper, we propose a new deep learning (DL) tool named Generative Artificial Intelligence Netlists SynthesIS (GAINESIS). GAINESIS is based on the Wasserstein Conditional Generative Adversarial Network (WCGAN) algorithm and area–power analysis features from the GLN phase and synthesizes new normal and infected circuit samples for this phase. Based on our GAINESIS tool, we synthesized new data sets, different in size, and developed and compared seven ML classifiers. The results demonstrate that our new generated data sets significantly enhance the performance of ML classifiers compared with the initial data set of Trust-HUB.

Keywords: artificial intelligence; generative learning; hardware trojan; application-specific integrated circuit; gate-level netlists; synthesis; new generated data sets; area–power features; GAN; WCGAN

Citation: Liakos, K.G.; Georgakilas, G.K.; Plessas, F.C.; Kitsos, P. GAINESIS: Generative Artificial Intelligence NEtlists SynthesIS. *Electronics* **2022**, *11*, 245. https://doi.org/10.3390/electronics11020245

Academic Editor: Esteban Tlelo-Cuautle

Received: 13 December 2021
Accepted: 11 January 2022
Published: 13 January 2022

1. Introduction

Every year, more and more innovative applications based on technology are developed and implemented in every aspect of our lives. The majority of these applications are based on Internet of Things (IoT) devices and artificial intelligence (AI), aiming to provide us with the ability to remotely access information and data from any device and automate tasks. However, all these technological breakthroughs do not come without disadvantages.

IoT devices consist mainly of sophisticated Application-Specific Integrated Circuit (ASIC)—Integrated Circuits (ICs). To reduce operating costs and facilitate mass production, design companies frequently outsource IC fabrication to third-party foundries. This process increases the risk of intrusion attacks in the form of hardware viruses, also known as hardware trojans (HTs). In the field of electronics, HT viruses are a critical problem that have the potential to become an outbreak in the coming years, presenting a significant threat both technologically and socially.

HTs are related to unwanted modifications to circuits that occur during the pre-silicon and post-silicon stages. Because of the complexity of modern circuits, HTs can be inserted at any phase of IC development and remain inactive until activated by a variety of activation mechanisms. HTs are related to total circuit collapse, unexpected IC failures and the leakage of sensitive information [1]. Therefore, developing well-designed and efficient HT countermeasures are of the utmost importance. The HT structure consists of an activation

mechanism (trigger) and an effect (payload) (Figure 1). HTs remain totally silent and via rare events or signals their triggers are activated [1], based on two logics, sequential or combinational. Sequential HTs need a sequence of rare signals for their activation, while the activation of combinational HTs is based on the simultaneous presence of a combination of rare signals. Furthermore, HT attacks are grouped into two categories of attacks, cryptographic engine and processor attacks. Cryptographic engine attacks try to leak encrypted information through various attack mechanisms, while the general-purpose processors aim to degrade or even to totally destroy the system via the memory, at lower levels of the processor and kernel.

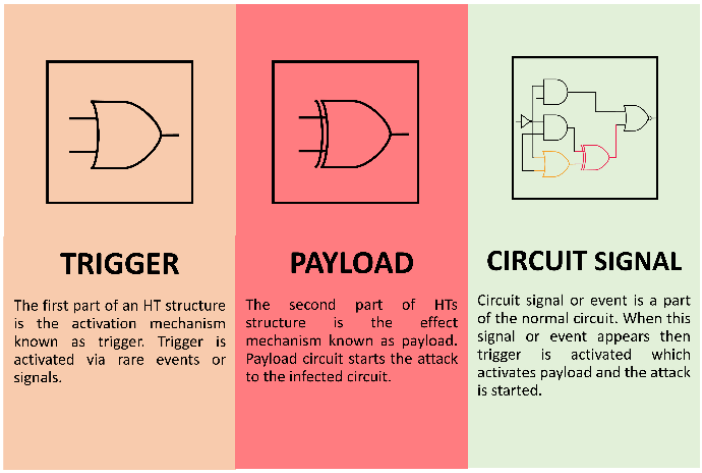

Figure 1. Hardware trojan structure.

Ideally, any unwanted alteration applied to an IC should be detected at any phase of the pre-silicon (e.g., Design Rule Checking—DRC, and Layout vs. Schematic—LVS checking.) and post-silicon verification stages. However, the pre- or post-silicon stage of an IC requires the IC golden model. This information is not always available, particularly for designs that are based on Ips that originate from mediator manufacturers. HT attacks can be divided according to the number of phases for each stage in the circuit's production chain at the register transfer level (RTL), GLN, placement and routing (P&R) and graphic database system II (GDSII) for the pre-silicon stage, as well as fabrication and testing–assembly for the post-silicon stage (Figure 2). Depending on the targeted phase, the attacker might obtain full access to design files and source code, or compromise computer-aided design tools and scripts to output a modified IC representation without altering the source code. Fabrication attacks, on the other hand, take place after tape-out and can remove or add components via layout geometry modification, reverse engineering or IC metering.

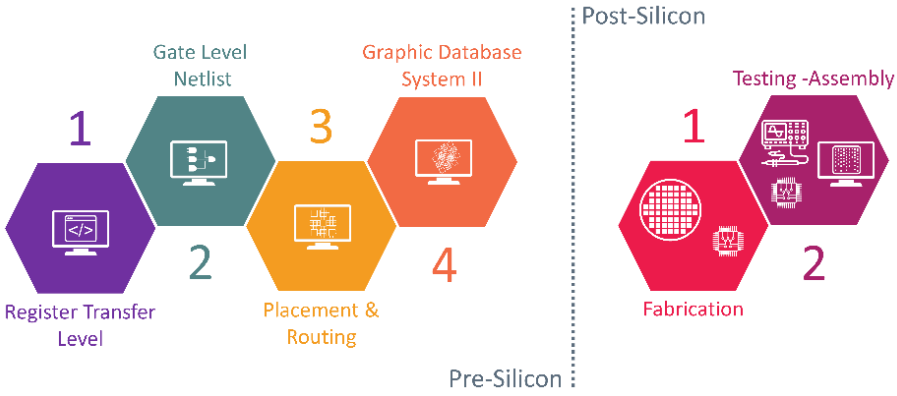

Figure 2. Overview of ASIC production chains.

Machine learning (ML) [2] and deep learning (DL) [3] in particular represent a collection of algorithms for modeling patterns embedded in data. DL has become very popular, especially in the last decade, for the development of solutions in multiple scientific fields, the industry, bioinformatics, agriculture, etc. [4,5]. In the hardware security field, a plethora of ML-based approaches for HT detection have been introduced in the last six years [6]. For the pre-silicon stage, these studies aim for the classification of normal and HT-infected circuits at the GLN phase, using area and power analysis GLN features such as number of gates, number of nets, number of multiplexers, number of flip-flops, number of cells and number of ports, as well as total, switching and combinational power. The most frequently used ML algorithms are support vector machine (SVM) and random forest (RF), with SVM typically ranking as the best-performing model [7–10].

Most studies in the field of HTs utilize the public Trust-HUB [11,12] library of circuit designs for extracting features related to both HT-free and HT-infected Ics. Utilizing the Trust-HUB repository has three major disadvantages: since the majority of circuits are designed for field-programmable gate arrays (FPGA), there is an imbalance between HT-free (N = 18) and HT-infected (N = 880) circuits, the circuits do not have diversity, and they are large in size, which means that they are easier to detect. The lack of HT-free and diversity designs leads to the creation of imbalanced data sets and subsequently to highly unreliable models with low generalization capacity which are incapable of detecting small-in-size HTs. It is becoming evident that the HT detection field requires a much higher number of circuit and diversity designs than what is already available in Trust-Hub, for developing robust ML models. This is not an easy task, since the majority of IC designs are protected by IP rights and will hardly ever be deposited in public repositories such as Trust-HUB. Thus, the community will have to become creative and make the most out of the available circuit designs from public resources.

This study aims to provide a solution to the Trust-HUB HT-free (TF) and HT-infected (TI) circuits imbalance problem, for the first time, by developing a feature generative approach based on Generative Adversarial Networks (GANs), named GAINESIS. GAINESIS utilizes a WCGAN model for the synthesis of new HT-free and HT-infected circuit features from the GLN phase. GANs are mostly used in the computer vision field for generating artificial images on various domains, such as realistic photographs of human faces [13], textual descriptions of birds and flowers [14], reconstructing damaged photographs of human faces [15], removing rain and snow from photographs [16] and many other functions. For the development of GAINESIS, the Design Compiler NXT tool [17] was utilized to synthesize 880 circuits (18 TF and 862 TI) at the GLN phase based on designs deposited in Trust-HUB. In-house-developed scripts were used to extract power and time features and to create the original data set. Multiple ML algorithms were tested on the original data set and the best-performing one (gradient boosting—GB) was used to further benchmark multiple GAN flavors and select the one that was better suited to the HT detection field (WCGAN). Based on the final GAINESIS model, new synthetic data sets of different sizes were generated and used to train distinct GB models to assess the applicability of GANs in the HT detection field.

2. Methodology

The development of GAINESIS was based on Python v3.6 [18] and all benchmarks were performed on an Intel X-Series I7-7740X computer system equipped with the NVIDIA GTX 1060 GPU. Tensorflow-GPU v1.3 [19], Keras v2.0 [20], Scikit-learn [21], the XGBoost library [22] and Jupyter Notebook [23] environment were used to develop all of the tested GAN and ML models.

2.1. Scheme of GAINESIS Methodology

Initially, all circuit benchmarks in Verilog format (1) were downloaded from the Trust-HUB repository. Design Compiler NXT and the FreePDK45nm open cell library were used to design the GLN phase of the circuits, a process also known as GLN synthesis (2).

Subsequently, in-house scripts were developed to generate and extract area, power and time analysis features for each of the designed GLN benchmarks (3). The initial real data set consisted of 880 samples, 18 TF and 862 TI, and the features utilized (N = 11) were number of ports, number of nets, number of cells, number of sequential cells, number of references (number of multiplexers and number of gates), net switching power, total dynamic power, combinational switching power, combinational total power, total switching power and total power (4). For the development of our initial real-data-based data set classifier we split our initial real data set into two sets, a training (80%, 704 samples) and a test (20%, 176 samples) set (5 and 6). The training of the seven ML-based classifiers was implemented based on the training set. Specifically, the seven models are based on seven algorithms, GB [24], k-nearest neighbors (KNN) [25], logistic regression (LR) [26], multilayer perceptron (MLP) [27], RF [28], SVM [29] and XGB [22]. It is worth mentioning that XGB was used for the first time for the classification of HTs at the GLN phase. For the development of each classifier, we used and combined a variety of hyperparameters to optimize each classifier (7). For our initial real data set we selected the best-performing classifier based on Precision, Sensitivity, Specificity and F1-score metrics (8), which was a GB-based classifier (9).

Next, we explored our real training data set and found that TI circuits have a larger area and consume more power compared with TF circuits (10). From the exploration of our real data set, it became evident that the Trust-HUB initial real data set is highly imbalanced. We postulated that GANs can be used to remedy this problem and provide arbitrary numbers of synthetic TF and TI feature vectors for training robust ML classifiers. Four GAN models were developed based on the vanilla GAN [30], conditional GAN (CGAN) [31], Wasserstein GAN (WGAN) [32], and WCGAN [33] algorithms. After the training of our four models, we optimized and evaluated them (11), and we picked the models with the best and the worst performances (12 and 13). Next, we synthesized new generated data sets based on our best and our worst-performing models (14 and 15). We combined the new generated data sets from our best and worst models with the initial real training data set to produce our mixed data sets (16 and 17).

Furthermore, we used all of the new data sets for the development and comparison of our new GB-based classifiers (18). For the development of the new GB-based classifiers, each of the data sets was split into two sets, a test (20%) set and a training (80%) set (19 and 20). Again, the training of the new GB-based classifiers was implemented based on the training sets (21) and their evaluation implemented based on the test sets. We selected as the new improvement GB-based classifier the best-performing classifier based on Precision, Sensitivity, Specificity and F1-score metrics (22), which was the GB-WCGAN-Mixed-600-based classifier (23).

Our final step was to compare our initial real GB-based classifier with our new best GB-WCGAN-Mixed-600-based classifier. Thus, we evaluated our GB-WCGAN-Mixed-600-based classifier with our initial real test set (24). Our scheme is illustrated in Figure 3. It needs to be mentioned that for the development of ML-based models, we used a 10-fold cross-validation process, which was repeated 50 times on each training set. The performance of the algorithms on the test set was implemented using a score cutoff of 0.5.

2.2. Initial Data Set Development

The process of data set development is the most critical step for the development of a robust ML model. In this instance, the data set should consist of circuits with diverse types, sizes and HT functions. We developed our data set by analyzing all benchmarks accessible in the Trust-HUB benchmark library, but we were not able to meet all the requirements of diversity in size and function through the lack of diversity in terms of the size and function of Trust-HUB benchmarks. Our first step was to design, with the Design Compiler NXT tool and FreePDK45nm circuit library [34], the TF and TI circuit benchmarks of Trust-HUB, which were in Verilog form. Next, with custom scripts we extracted area, power and time features from the design analysis produced from the Compiler NXT tool. The initial

extracted features were 51 in number, but many produced zero or not available feature values. So, we cleaned our data set of these features and prepared it for the development of our method. As a result, our data set consisted of 11 features: five area and six power analysis features (Table 1). Specifically, the five area features were the number of ports, nets, cells, and sequential cells, as well as the number of gates and multiplexers, or according to the Design Compiler NXT the number of references, which is how we report it in this study. The six power features were the net switching power, combinational switching power, total switching power, total dynamic power, combinational total power, and total power of each designed circuit. So, our initial real data set consisted of a total of 880 designed circuits. From the 880 circuits, 18 were normal or TF circuits which consisted of positive samples with a class label equal to one (label = 1). The 862 were modified normal circuits infected with HTs or TI, which consisted of negative samples with a class label equal to zero (label = 0). It must be mentioned that we named our initial real data set the REAL-880 data set. So, our initial REAL-880 data set consisted of a total of 880 designed samples. From the 880 samples, 18 were TF and 862 were TI. For the training, we used 704 samples, 14 TF and 690 TI (80%), and for the evaluation 176 samples, 4 TF and 172 TI (20%).

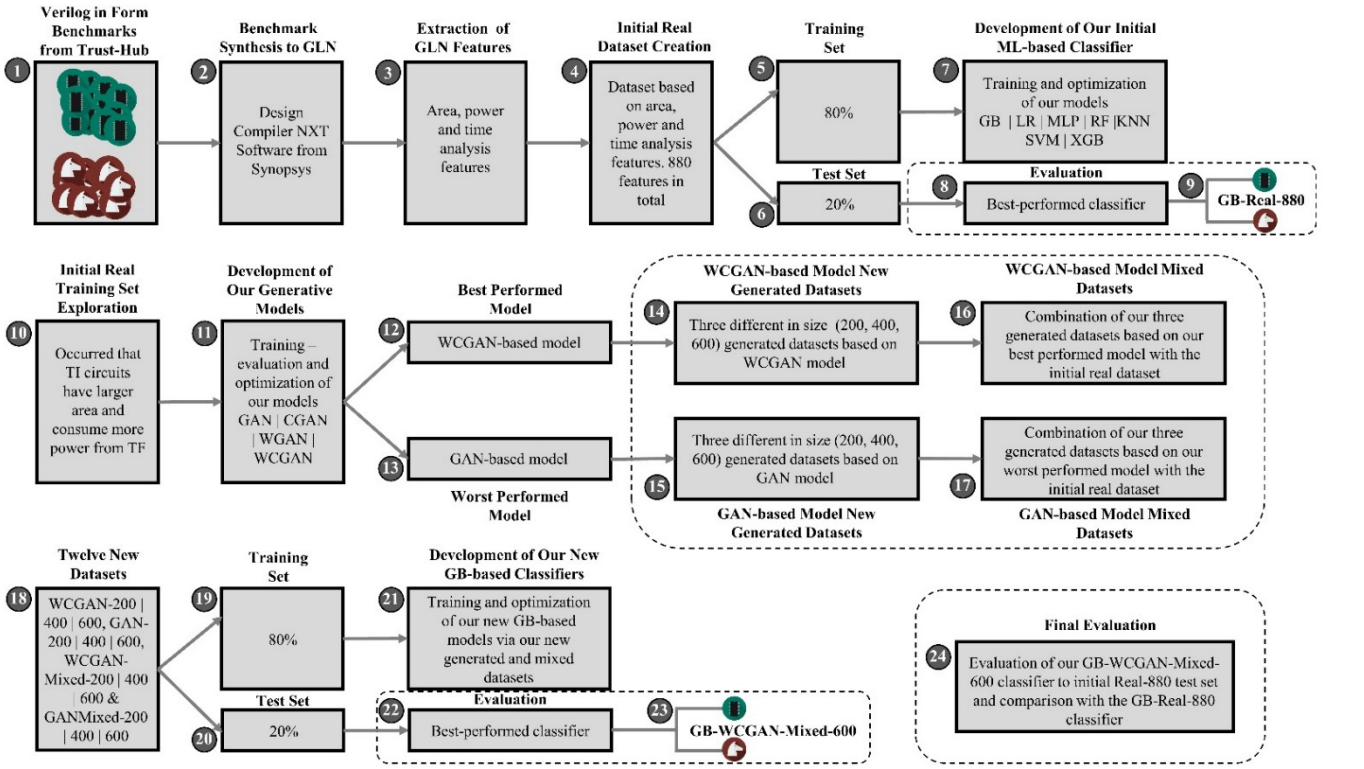

Figure 3. Scheme of our Artificial Intelligence-based approach for safeguarding integrated circuits at gate-level netlist phase against hardware Trojans, GAINESIS.

2.3. Machine Learning Classifiers Development

To be able to develop our ML-based classifier for our REAL-880 data set we trained and optimized seven ML-based classification models. We based this method on our previous work [10] where we developed six ML-based classifiers for the classification of TF and TI circuits via area and power analysis features in the GLN phase, with the difference that in this study we used an extra ML algorithm, the XGB. It must be mentioned that for the training and optimization of each classifier we used a combination of the appropriate hyperparameters based on each ML-based algorithm, which consisted of a wide range of values. The values given in each parameter were related to the type and size of the features of the training set, as well as to the computing power of our system.

Table 1. Table with our eleven area and power analysis features.

Analysis	Feature
Area	Number of ports
	Number of nets
	Number of cells
	Number of sequential cells
	Number of references
Power	Net switching power
	Total dynamic power
	Combinational switching power
	Combinational total power
	Total switching power
	Total power

2.3.1. GB-Based Classifier

GB-based classifier development is based on the combination of four hyperparameters: learning rate, max tree depth, number of estimators and max features. The hyperparameter learning rate controls the gradient descent by evaluating the contribution of each tree to the final result. For the training of our GB-based classifier we used a list of learning rate values from 0.05 to 1. The number of estimators hyperparameter represents the total number of sequential trees to be modeled. We used a list of the number of estimators, with values from 10 to 100. The max tree depth hyperparameter controls the depth of the individual trees. We used a list of max tree depth values from 1 to 10. Furthermore, the max features parameter represents the number of features that will be used for the best split. A list of max features values from 1 to 11 was used. The best combination of hyperparameters for our REAL-880 data set was: learning rate 0.05, number of estimators 10, max tree depth 11 and max features 10.

2.3.2. KNN-Based Classifier

For the development of our KNN-based classifier, we used five hyperparameters: number of neighbors, distances, leaf size, weights and metrics. The number of neighbors hyperparameter is the core deciding factor. For this hyperparameter, we used a list of values from 1 to 60. Distances were used in order for the KNN classifier to be able to calculate the distances between the point and points in the training set. On this occasion, we used a list of distance values from 1 to 10. The leaf size parameter defines the maximum number of points a node can hold. We used a list of leaf size values from 1 to 50. The weights parameter gives more weight to the points which are nearby and less weight to the points which are farther away. The uniform and distance weights were used for the training and optimization of our KNN-based model. The best combination of hyperparameters for our REAL-880 data set was: number of neighbors 1, distances 1, leaf size 1, weights 'uniform' and metric 'minkowski'.

2.3.3. LR-Based Classifier

For the training and optimization of our LR-based classifier, we used four hyperparameters: solver, penalty, C and max iterations. The solver hyperparameter solves optimization problems of the LR algorithm through coordinate descent (CD) algorithms. For this parameter, we used Newton-CG [35], limited-memory Broyden–Fletcher–Goldfarb–Shanno (LM-BFGS) [36], library large-scale linear LIBLINEAR [37], stochastic average gradient (SAG) [38] and SAGA [39] CD algorithms. Penalties were used to shrink the coefficients of the less contributed variable toward zero. We used three types of penalties: l1, l2 and elasticnet. The C parameter controls the penalty strength; we used a list of C values from 0.01 to 1000. The max iterations parameter is the maximum number of iterations taken for the solvers to converge. A list of max iterations values from 100 to 7000 was used. The

best combination of hyperparameters for our REAL-880 data set was: solver 'Newton-CG', penalty 'l2', C 0.01 and max iterations 100.

2.3.4. MLP-Based Classifier

For the training optimization of our MLP-based classifier, six hyperparameters were used: hidden layer sizes, activation, solver, alpha, max iterations and learning rate. The hidden layer sizes parameter defines the number of hidden layers of the network. A list of hidden layer size values from 10 to 50 was used. The activation function parameter was used to introduce non-linearity into the output of a neuron. A neural network has neurons that work in correspondence to weight, bias and their respective activation function. We used four types of activation function: identity, logistic, Tanh and ReLU. The solver parameter represents a stochastic gradient descent-based optimizer for optimizing the parameters in the computation graph. We used LM-BFGS, SGD and Adam optimizer. Alpha is a parameter for the regularization term, which combats overfitting by constraining the size of the weights. A list of alpha values from 0.001 to 0.9 was used. The maximum number of iterations parameter determines the solver. The solver iterates to this number of maximum iterations. A list of 100–1000 values from the maximum number of iterations was used. The learning rate parameter controls the rate of speed at which the model learns. We used three types of learning rate: constant, adaptive and invscaling. The best combination of hyperparameters for our REAL-880 data set was: hidden layer sizes 30, 30, 30, activation 'ReLU', solver 'Adam', alpha 0.0001, max iterations 500 and learning rate 'constant'.

2.3.5. RF-Based Classifier

For our RF-based classifier training optimization, we used four hyperparameters: number of estimators, max features, max depth and min sample leaf. The number of estimators parameter defines the number of trees in the algorithm. We used a list of the number of estimator parameter values from 100 to 5000. The max features parameter defines the number of features to consider when looking for the best split. We used auto, sqrt and log2 max feature values. The max depth parameter represents the depth of each tree in the forest. The deeper the tree, the more splits it has, and it collects more information about the data. A list of max depth values from 2 to 50 was used. The min sample leaf parameter consists of the minimum number of samples required to be at a leaf node. We used values from 1 to 20 for this parameter. The best combination of hyperparameters for our REAL-880 data set was: number of estimators 200, max features 'auto', max depth 10 and min sample leaf 2.

2.3.6. SVM-Based Classifier

We trained and optimized our SVM-based classifier according to three hyperparameters: C, gamma and kernel. The C parameter is a regularization parameter. It controls the tradeoff between the smooth decision boundary and classifying the training points correctly. C values from 0.0001 to 100 were used. The gamma parameter defines how far the influence of a single training example reaches. We used scale and auto gamma values. The kernel parameter specifies the kernel type to be used in the algorithm to improve the classification accuracy of the classifier. We used four types of kernels: linear, polynomial, gaussian radial basis function (RDF) and sigmoid. The best combination of hyperparameters for our REAL-880 data set was: C 20, gamma 'scale' and kernel 'poly'.

2.3.7. XGB-Based Classifier

For the training and optimization of our XGB-based classifier we used three hyperparameters: learning rate, number of estimators and max depth. The learning rate parameter controls the gradient descent. We used a list of learning rate values from 0.05 to 1. The number of estimators hyperparameter represents the total number of sequential trees to be modeled. We used a list of the number of estimators values from 10 to 100. The max tree depth hyperparameter controls the depth of the individual trees. We used a list of max tree

depth values from 1 to 11. Furthermore, the max features parameter represents the number of features that will be used for the best split. A list of max features values from 1 to 11 was used. The best combination of hyperparameters for our REAL-880 data set was: learning rate 0.25, number of estimators 60 and max depth 5.

2.4. Machine Learning Classifiers Evaluation

In this study, for the evaluation of the performance of ML algorithms we used Accuracy, Precision, Recall or Sensitivity, Specificity, 1 − Specificity and F1-score metrics. To evaluate the mentioned metrics, we used the values True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN). The TP value represents the number of TI circuits classified as TI, while the FP value represents the number of TF circuits that are wrongly classified as TI. On the other hand, the FN value represents the TI circuits that are classified as TF, and the TN value represents the number of TF circuits classified as TF. These values are used for the calculation of Accuracy (1), Precision (2), Recall (3), Specificity (4), 1 − Specificity (5) and F1 (6) metrics. As mentioned, positive samples indicate the TI circuits and our negative samples indicate the TF circuits. Accuracy is defined as the number of correct predictions divided by the total number of predictions (1). Precision defines the total number of TP values divided by the total number of all positive values (2). Recall defines the total number of TP values divided by the total number of TP and FN values (3) and can be characterized as the True Positive Rate (TPR). Specificity defines the total number of TN values divided by the total number of TN and FP values (4) and can be characterized as the True Negative Rate (TNR). 1 − Specificity defines the total number of FP values divided by the total number of TN and FP values (5). F1-score is the harmonic mean of Precision and Recall and is defined from the multiplication of Precision by Recall and then by the number two divided by the product of Precision and Recall (6). Additionally, based on these metrics we produced the receiver operating characteristic (ROC) and Precision–Recall curves. The ROC curve calculates the area under the curve (AUC) which is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve, while average precision (AP) summarizes a Precision–Recall curve as the weighted mean of the precisions achieved at each threshold (7).

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (1)$$

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \quad (2)$$

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \quad (3)$$

$$\text{Specificity} = \text{TN}/(\text{TN} + \text{FP}) \quad (4)$$

$$1 - \text{Specificity} = \text{FP}/(\text{TN} + \text{FP}) \quad (5)$$

$$\text{F1-score} = 2(\text{Precision} * \text{Sensitivity})/(\text{Precision} + \text{Sensitivity}) \quad (6)$$

$$\text{AP} = \sum_n [(\text{Rn} - \text{R}(\text{n} - 1)) * \text{Pn}] \quad (7)$$

From Figure 4, it can be observed that for the training set all classifiers had a good performance. On the other hand, for the test evaluation set, none of our classifiers performed well. Specifically, the GB-based classifier was found to be the best-performing classifier on the test set compared with the other six, with 97.72% Accuracy, 74.13% Precision, 62.20% Recall and 66.08% F1-score (Figure 5). Additionally, good results were returned for MLP-based classifier, with 96.59% Accuracy, 61.92% Precision, 61.62% Recall and 61.62% and F1-score 61.92%. Thus, according to the results, the GB-based classifier was the most efficient. Based on the GB algorithm, we developed and compared our real and our new generated data sets.

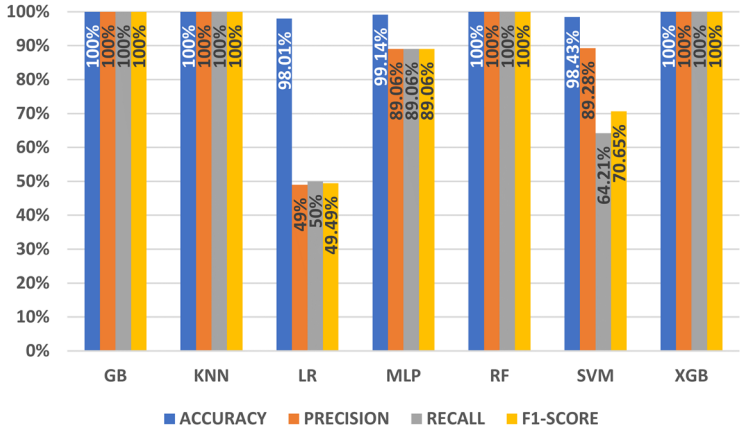

Figure 4. Histograms of the performance of our seven ML-based classifiers on our REAL-880 training set.

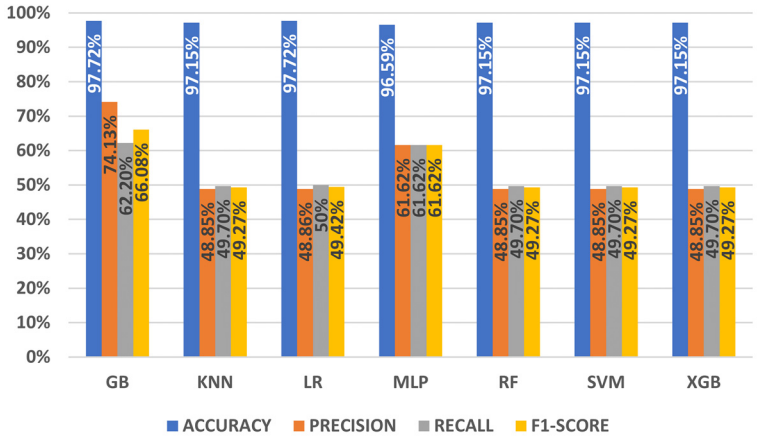

Figure 5. Histograms of the performance of our seven ML models on our REAL-880 test set.

2.5. GAINESIS Development

Our first step for the development of our new synthetic data sets based on our generative models was to explore our real training data set. As previously mentioned, the TI class consists of 98% of our total initial real training data set, with 704 samples, and the TF class only 2%, with 14 samples. From the exploration of our real training data set, we observed that TI samples in their majority had greater mean values compared with TF. This is logical, because TI circuits are modified TF circuits with HTs and use extra area features such as gates, cells, nets, etc., for the construction of the structure of the inserted HT. On the other hand, these extra area features need more power. Thus, TI circuits consume more power from TF (Figure 6).

As it turns out, our real training data set is inadequate and unequal. The data are the most significant part of any ML project. A lack of data samples and lack of diversity data can lead to mediocre ML projects. Additionally, supervised learning models require data, and their performance is largely based on the size of the training data available. So, to solve these functional problems, we needed to produce more TF samples. In the bibliography exist different techniques for data synthesis on ML. In our study, we used a novel state-of-the-art technique for data-synthesis-based DL, known as GANs. GANs algorithms mainly are used for the field of computer vision and especially for image editing and data generation, and use 2D or 3D (two- or three-dimensional) networks based on convolutional neural networks (CNNs). In this study, we modified the networks to 1D (one-dimensional) networks based on deep neural networks (DNNs), because our data set consisted of 1D features.

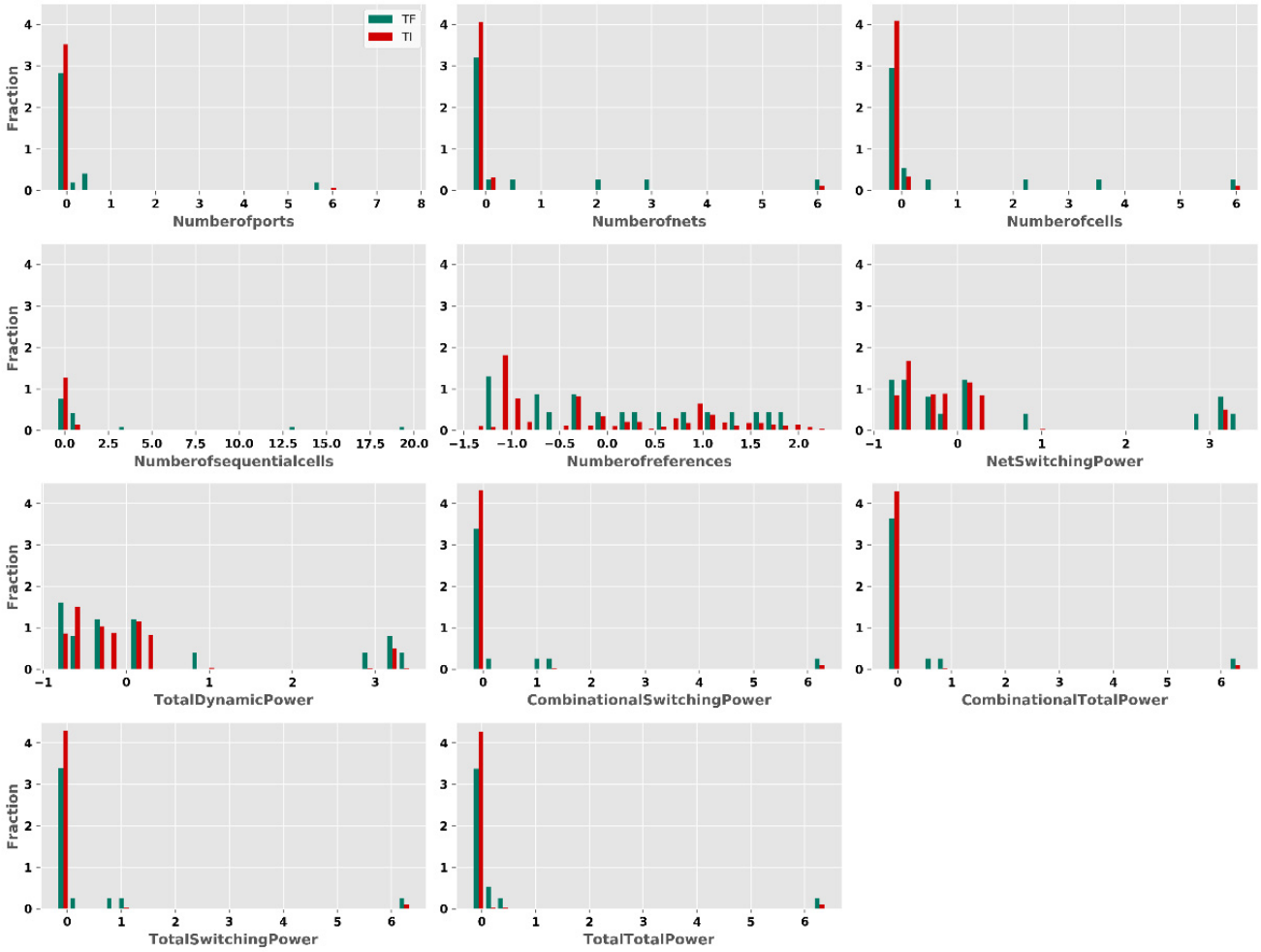

Figure 6. Data distributions by feature and class.

To solve these functional problems which occurred from the lack of data samples, we developed and compared four generative learning models. As mentioned, we developed four models based on four different algorithms, GAN, CGAN, WGAN and WCGAN, for the synthesis of new samples. Specifically, GANs consists of two models, a generator and a discriminator. These are trained simultaneously by an adversarial process. The generator learns to produce data that look real based on real samples, while the discriminator learns to distinguish the real from generated data to the point where it is no longer able to distinguish them. CGANs is an architecture close to the original GANs, with the only difference being that it makes use of the class labels feature. CGAN, with the use of the class labels feature, allows the targeted synthesis of a given sample. WGANs are based on the philosophy of GANs, with the difference that they use the Wasserstein distance metric for the development of the two models, generator and discriminator. The Wasserstein distance metric provides a meaningful and smooth representation of the distance between distributions. This algorithm enhances model stability during training and gives a loss function that corresponds with sample quality. The last algorithm which was used and compared for the generation of new samples was the WCGAN. WCGANs have the same functionality as the WGANs, with the difference that the CGANs make use of the class labels feature for the training of the generator and discriminator models. Next, the hyperparameters that were used to improve the development of our four models are presented.

For the development of our four models, we used and combined six hyperparameters: learning rate, batch size, number of epochs, optimizer, number of units in a dense layer and activation function. Each hyperparameter contained a wide range of values. The

hyperparameter learning rate controls the model in response to the estimated error each time the model weights are updated. On this occasion, we used a list of learning rate values from 0.0001 to 0.001. The hyperparameter batch size defines the number of samples that will be propagated through the network. We used a list of batch size values from 16 to 64. The number of epochs hyperparameter specifies the time in which the learning algorithm will process the whole training data set. We used different number of epochs values from 1000 to 50,000. The optimizer hyperparameter affects the attributes of the neural network such as weights and learning rate to reduce the losses. For the development of our models, we used three optimizers: stochastic gradient descent (SGD) [40], Adam [41] and root mean square propagation (RMSprop) [42]. The number of units in a dense layer hyperparameter affects the effectiveness of our models. On this occasion, we used different numbers of units in a dense layer, from 25 to 512. The activation function hyperparameter describes how the weighted sum of the input is turned into an output from a node or nodes in a network layer. Specifically, we used three activation functions: rectified linear unit (ReLU) [43], sigmoid [44] and hyperbolic tangent (Tanh) [45] (Table 2).

Table 2. Table with the range of hyperparameters for the generative learning models.

Hyperparameter	Range
Learning rate	0.0001–0.001
Batch size	16–64
Number of epochs	1000–50,000
Optimizers	SGD, Adam, RMSprop
Dense layer	25–512
Activation function	ReLU, sigmoid, Tanh

So, as mentioned for the development of our four models we combined all the values of each hyperparameter. The optimum hyperparameters combination was learning rate equal to 0.0005, batch size equal to 64, number of epochs equal to 50,000, optimizer being Adam, number of units in a dense layer equal to 128 for the first layer, and activation function being ReLU. It should be noted that for the development of the generator network for each layer, we multiplied exponentially by the number two the number of units in a dense layer, for the discriminator network we multiplied by number four the first dense layer, and for the other layers we divided it by the number two. Additionally, for the first three dense layers of the generator, the best activation function was ReLU, the same as for the discriminator, except for the last fourth dense layer of the discriminator network, in which the best activation function was sigmoid. The values given in each parameter were related to the type and size of the features of the training set, as well as to the ability of the computing power of our system.

In Tables 3 and 4 is presented the generator network for each of our four models. GAN- and WGAN-based models are different from CGAN and WCGAN because, as previously mentioned, CGAN- and WCGAN-based models use as an extra feature the class of the sample. Additionally, in Tables 5 and 6 is presented the discriminator network for each of our four models. The only difference between our models is in the input layer, because CGAN- and WCGAN-based models, as previously mentioned, use as an extra feature the class of the sample.

Table 3. GAN and WGAN models generator network.

Layer	Output	Parameters
Input layer 1	(None, 11)	0
Dense 1	(None, 128)	1536
Dense 2	(None, 256)	33,024
Dense 3	(None, 512)	131,584
Dense 4	(None, 11)	5643

Table 4. CGAN and WCGAN models generator network.

Layer	Output	Parameters
Input layer 1	(None, 11)	0
Input layer 2	(None, 1)	0
Concatenate 1	(None, 12)	0
Dense 1	(None, 128)	1664
Dense 2	(None, 256)	33,024
Dense 3	(None, 512)	131,584
Dense 4	(None, 11)	5643
Concatenate 1	(None, 12)	0

Table 5. GAN and WGAN models discriminator network.

Layer	Output	Parameters
Input layer 1	(None, 11)	0
Dense 1	(None, 512)	6144
Dense 2	(None, 256)	131,328
Dense 3	(None, 128)	32,896
Dense 4	(None, 1)	129

Table 6. CGAN and WCGAN models discriminator network.

Layer	Output	Parameters
Input layer 1	(None, 12)	0
Dense 1	(None, 512)	6656
Dense 2	(None, 256)	131,328
Dense 3	(None, 128)	32,896
Dense 4	(None, 1)	129

2.6. GAINESIS Evaluation

To evaluate the performance of our models, we used metrics such as the Minmax and Wasserstein loss functions. Specifically, the Minmax loss function reflects the distance between the distribution of the generated data and the distribution of the real data, for GAN- and CGAN-based models. GAN and CGAN algorithms use two Minmax loss functions, one for the generator and one for the discriminator. A single measure of distance between probability distributions yields both generator and discriminator losses. The generator can only change one component of the distance measure in any of these schemes, the term that represents the distribution of the fake. As a consequence, we eliminate the other term that reflects the distribution of the actual data during generator training. The formula for minmax loss is presented in Equation (8). D(x) estimates the probability that the real data instance x is real for the discriminator. Ex is the expected value over all real data instances. G(z) is the output of the generator when given noise z. D(G(z)) estimates the probability that a fake instance is real for the discriminator. Ez is the expected value over all generated fake instances G(z)). For the evaluation of a model in WGAN and WCGAN algorithms, the discriminator does not classify instances but outputs a number. The discriminator aims to increase the output for real instances rather than fake instances. For this reason, we use the Wasserstein Discriminator Loss (9) and Generator Loss (10). Specifically, D(x) is the output for a real instance at the discriminator. G(z) is the output when given noise z, at the generator. D(G(z)) is the output for a fake instance at the discriminator.

$$\text{Minmax Loss} = E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))] \quad (8)$$

$$\text{Wasserstein Discriminator Loss} = D(x) - D(G(z)) \quad (9)$$

$$\text{Wasserstein Generator Loss} = D(G(z)) \quad (10)$$

From our four generative learning models, our WCGAN-based model was found to be the best-performing model in epoch 47,000 of 50,000 epochs, with a generator loss value equal to 0.102 (Figure 7) and discriminator loss value equal to 0.0984 (Figure 8). The next best-performing model was our WGAN-based model for epoch 47,000 from 50,000 epochs, with a generator loss value equal to 0.0995 (Figure 7) and discriminator loss value equal to 0.114 (Figure 8), while our CGAN-based model's best epoch was 48,000 from 50,000 epochs, with a generator loss value equal to 0.369 (Figure 7) and discriminator loss value equal to 0.263 (Figure 8). Our GAN-based model was our worst-performing model, with the best epoch being epoch 46,000 of 50,000 epochs, and a generator loss value equal to 0.453 (Figure 7) and discriminator loss equal to 0.273 (Figure 8). Additionally, we displayed for each epoch the ability of each model to synthesize new generated samples based on real samples according to the most important features. From this, we observed that our best-performing WCGAN-based model (Figure 9) was able to synthesize better-generated samples compared with the other models and especially compared with our worst-performing GAN-based model (Figure 10). To distinguish any differences in the quality of the new generated samples and to confirm the evaluation of our models, we synthesized new samples based on our best-performing WCGAN-based model and based on our worst-performing model GAN-based model in order to develop new GB-based classifiers.

2.7. Synthesis of New Generated Data Sets

After we finished with the training, optimization, and evaluation of our four models, we selected the best- and worst-performing models. As occurred previously, the model that learned to synthesize new generated data similar to the real data was our WCGAN-based model, while the model with the worst performance was our GAN-based model. Additionally, to be able to observe any differences, we created differently sized data sets from each model. As a result, our new generated data sets, which are based on our best-performing WCGAN model, were named WCGAN-200, WCGAN-400 and WCGAN-600 according to the size of the sample. Additionally, our new generated data sets were based on our worst-performing GAN model, and named GAN-200, GAN-400 and GAN-600. Next, we mixed each new generated data set with the initial real training data set, not the test set, to be able to evaluate our best new GB-based classifier in the real test data set. So, our mixed data sets were WCGAN-Mixed-200, WCGAN-Mixed-400, WCGAN-Mixed-600, GAN-Mixed-200, GAN-Mixed-400, and GAN-mixed-600. In total, we had 12 new data sets to compare. As in the real data set, the new generated data sets' TF circuits consisted of positive samples, with a class label equal to one (label = 1), and TI circuits consisted of our negative samples, with a class label equal to zero (label = 0). Additionally, as mentioned previously, 80% of each data set was used for the training of our new GB-based models, and 20% for the evaluation. Next, we analyzed the details of each data set and how these were used for the training and evaluation of our new GB-based models.

As a result, our new generated data sets were six in total, three for each model and three data sets different in sample size. WCGAN-200 and GAN-200 data sets were our data sets smallest in sample size and consisted of 432 samples, 216 TF and 216 TI samples. A total of 345 samples, 171 TF and 174 TI, were used for the training, and 87 samples, 45 TF and 42 TI, were used for the evaluation of our new GB-based models. Our middle range data sets were WCGAN-400 and GAN-400 data sets. They consisted of 864 samples: 432 TF and 432 TI samples. A total of 691 samples, 357 TF and 334 TI samples, was used for the training and 173 samples, 75 TF and 98 TI samples, for the evaluation of our new GB-based classifiers. Our large-sample generated data sets were WCGAN-600 and GAN-600. These data sets consisted of 1296 samples, 648 TF and 648 TI samples. For the training of our new GB-based classifiers, we used 1036 samples, 523 TF and 513 TI, and for the evaluation 260 samples, 125 TF and 135 TI. Furthermore, as well as our new generated data sets, our mixed data sets were in total six in number. WCGAN-Mixed-200 and GAN-Mixed-200 data sets each consisted of one in total from 1136 samples, 230 TF and 906 TI. From these

908 samples, 191 TF and 717 TI were used for the training and 228 samples, 43 TF and 185 TI, were used for the evaluation. WCGAN-Mixed-400 and GAN-Mixed-400 data sets consisted, respectively, of 1568 samples in total, 446 TF and 1122 TI. From these 1254 samples, 359 TF and 895 TI were used as a training set and 314 samples, 91 TF and 223 TI samples were used as an evaluation set. Our last mixed data sets were WCGAN-Mixed-600 and GAN-Mixed-600. Each one of these data sets had in total 2000 samples, 662 TF and 1338 TI samples. The training set consisted of 1600 samples, 544 TF and 1056 TI samples while the evaluation set consisted of 400 samples, 122 TF and 278 TI (Figure 11). In the next section is presented how we trained and optimized our seven classification models.

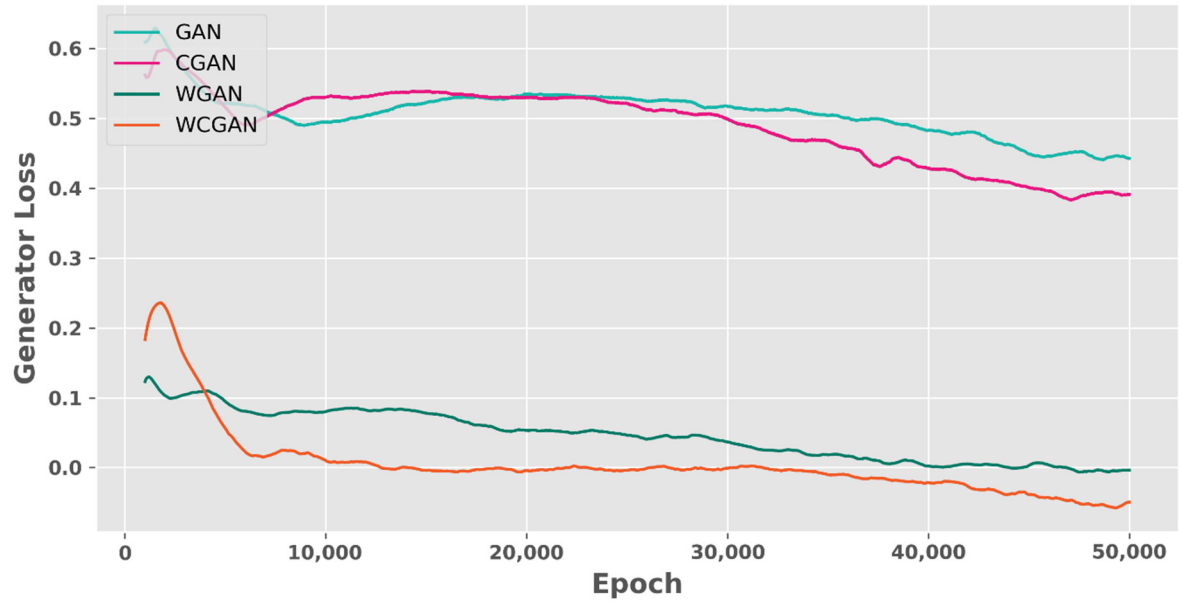

Figure 7. Generator loss values of our four models for each epoch.

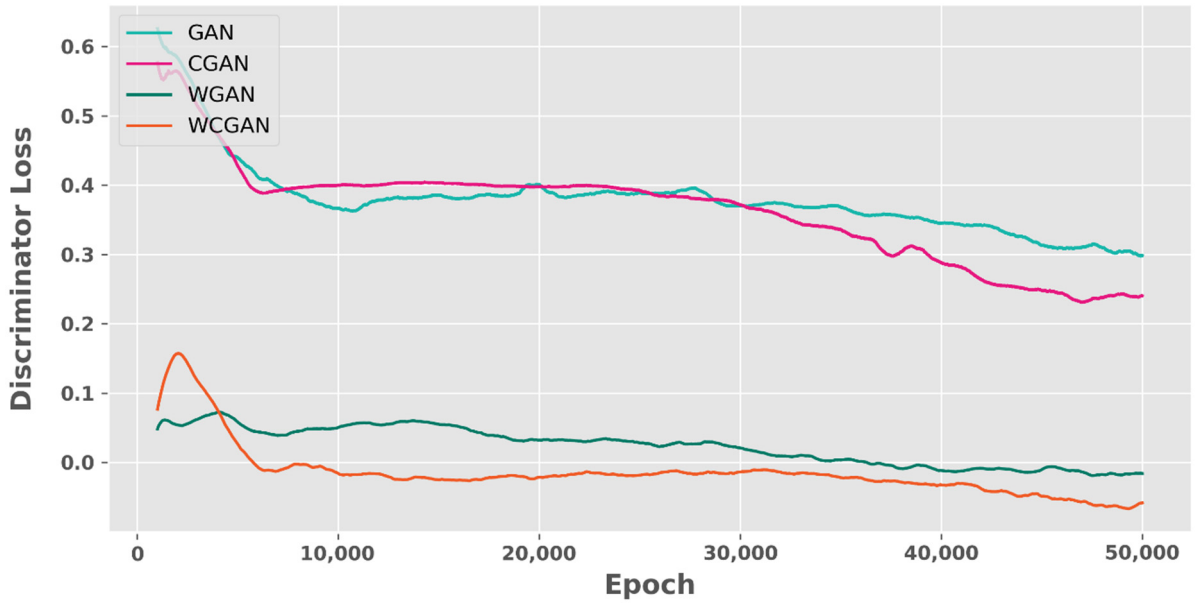

Figure 8. Discriminator loss values of our four models for each epoch.

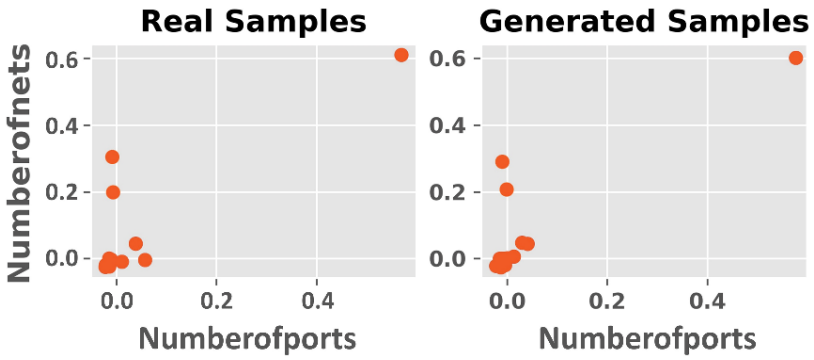

Figure 9. Presentation of how our best-performing WCGAN-based model learned to synthesize new generated samples based on real samples.

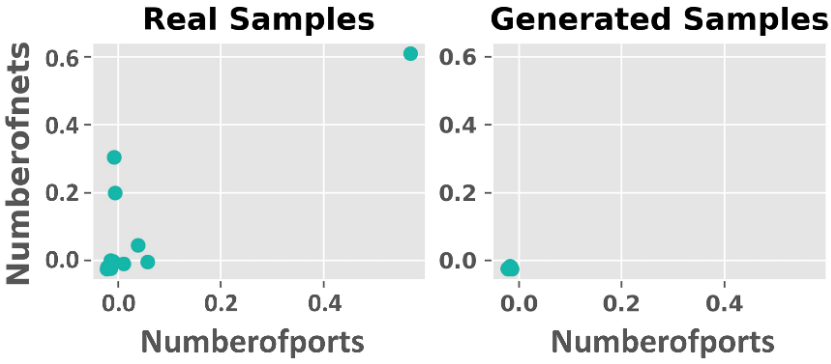

Figure 10. Presentation of how our worst-performing GAN-based model learned to synthesize new generated samples based on real samples.

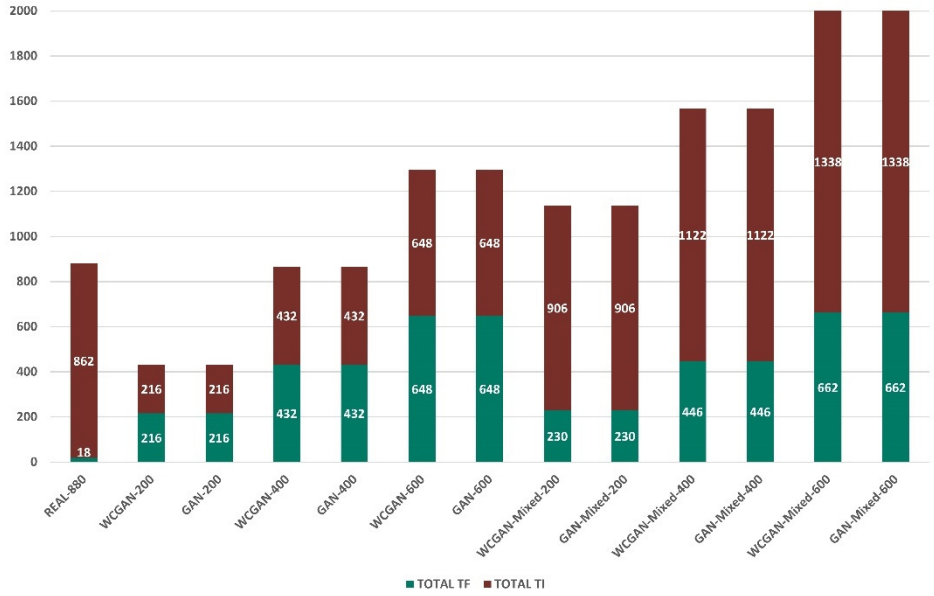

Figure 11. Histograms with the distribution of TF and TI samples for our 13 data sets.

3. Results

3.1. New Generated Data Sets Results

Our first step was to compare our six new generated data sets. So, we developed six new GB-based classifiers, one for each data set. According to Figures 12 and 13, both for the training and the evaluation phase, our WCGAN-based data sets enhanced even a little

the performance of the classifiers compared with our GAN-based data sets. Specifically, the GB-based classifiers for the evaluation phase obtained a 99.6% F1-score for our WCGAN-200 data set, 99.86% F1-score for our WCGAN-400 data set and 99.94% F1-score for our WCGAN-600 data set, while for our GAN-200 data set was obtained a 98.37% F1-score, 99.2% F1-score for our GAN-400 data set and 99.49% F1-score for our GAN-600 data set. Additionally, from the above, it can be observed that the performance of the classifiers was affected, and also by the size of the data set. Specifically, the data sets with more samples enhanced the performance of the classifier compared with the data sets with fewer samples, for both WCGAN-based and GAN-based data sets.

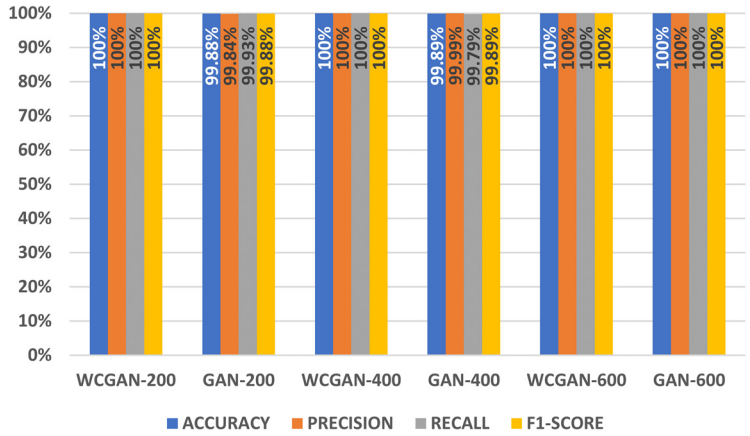

Figure 12. Histograms of the performance of our new GB-based classifiers on our new generated training sets.

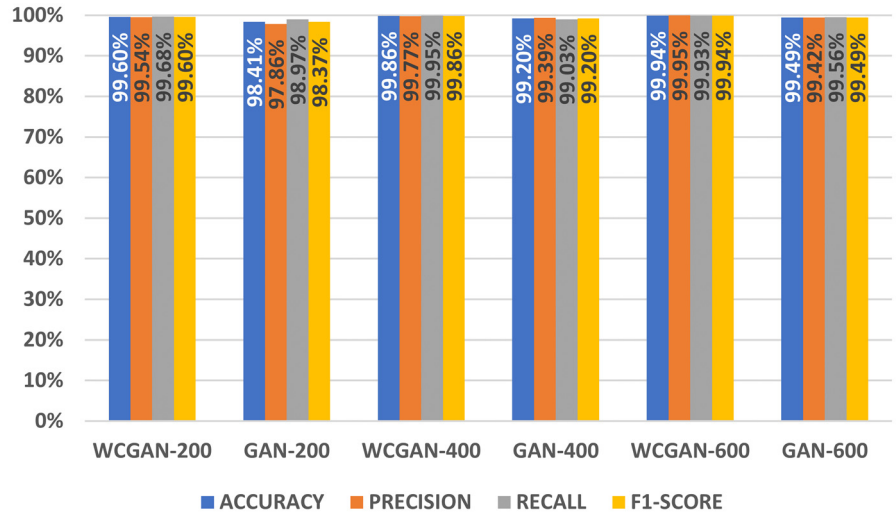

Figure 13. Histograms of the performance of our new GB-based classifiers on our new generated test sets.

3.2. Mixed Data Sets Results

Our next step was to compare our six mixed data sets. As previously mentioned, mixed data sets consisted of the new generated samples from our WCGAN-based and GAN-based generative models, respectively, and the initial real training data samples from our REAL-880 data set. According to Figures 14 and 15 emerged the same conclusions as in the comparison of the new generated data sets. Our best GB-classifier was the classifier that was developed based on the WCGAN-Mixed-600 data set. Specifically, our new mixed GB-based classifiers for the evaluation phase achieved a 95.08% F1-score for our WCGAN-Mixed-200 data set, 97.39% F1-score for our WCGAN-Mixed-400 data set and

98.26% F1-score for our WCGAN-Mixed-600 data set, while for our GAN-Mixed-200 data set was obtained a 94.59% F1-score, 97.61% F1-score for our GAN-Mixed-400 data set and 98.11% F1-score for our GAN-Mixed-600 data set.

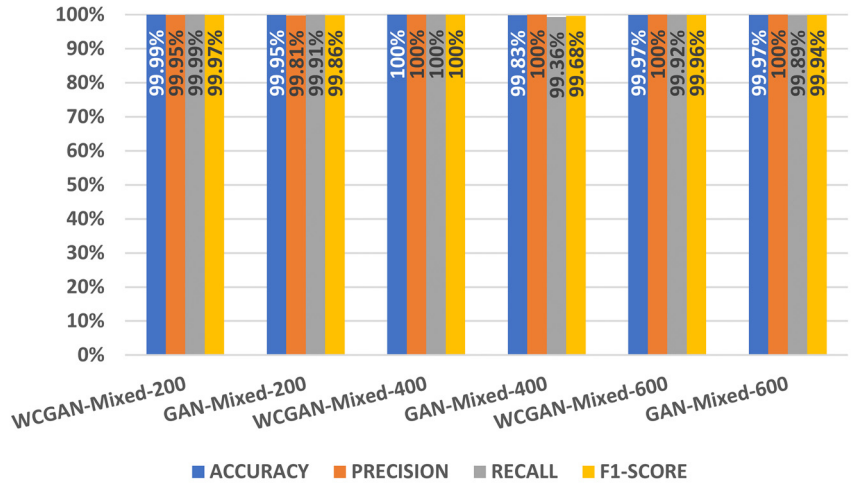

Figure 14. Histograms of the performance of our new GB-based classifiers on our mixed training sets.

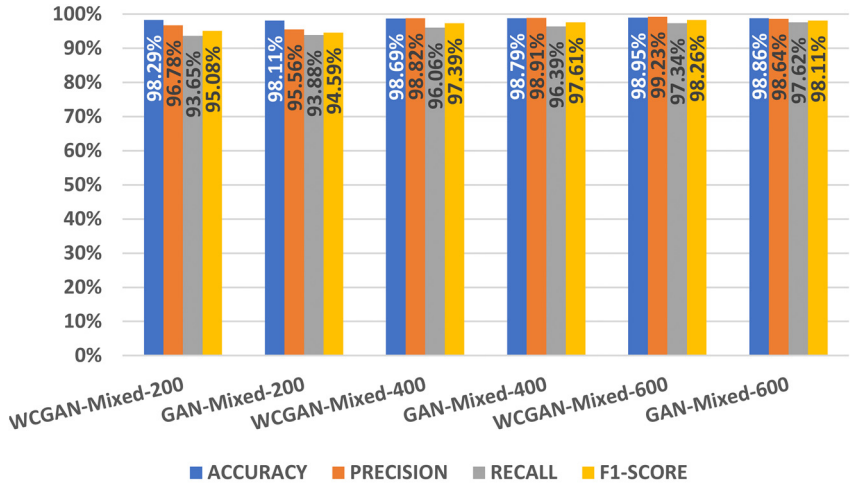

Figure 15. Histograms of the performance of our new GB-based classifiers on our mixed test sets.

3.3. All Data Sets Results

According to our results, our best new classifiers are based on WCGAN-Mixed-600 and GAN-Mixed-600 data sets. These newly generated data sets, in combination with our real training data set, managed to increase the F1-score for our new best-performing GB-based classifiers by 32.18% and 32.03%, respectively (Figure 16).

To be able to distinguish extra details between the WCGAN-Mixed-600 and GAN-Mixed-600 data sets we used ROC and Precision–Recall curves. Each GB-based classifier of each data set was tested with the test sets of each other. According to Figure 17, it can be observed that our WCGAN-600 (Figure 17c,d) and WCGAN-Mixed-600 (Figure 17g,h) data sets significantly enhanced the classification procedure compared with our GAN-600 (Figure 17e,f) and GAN-Mixed-600 data sets (Figure 17i,j). Specifically, our GB-WCGAN-Mixed-600 classifier, compared with the GB-GAN-Mixed-600 classifier, was able to classify with better performance 99% AUC and 99% AP for not only the GAN-Mixed-600 data set but also the REAL-880 data set, with 75% AUC and 16% AP compared with the GB-CGAN-Mixed-600 classifier, which obtained 70% AUC and 41% AP for the WCGAN-Mixed-600 data set and 68% AUC and only 9% AP for the REAL-880 data set. So, our new best classifier was the GB-WCGAN-Mixed-600.

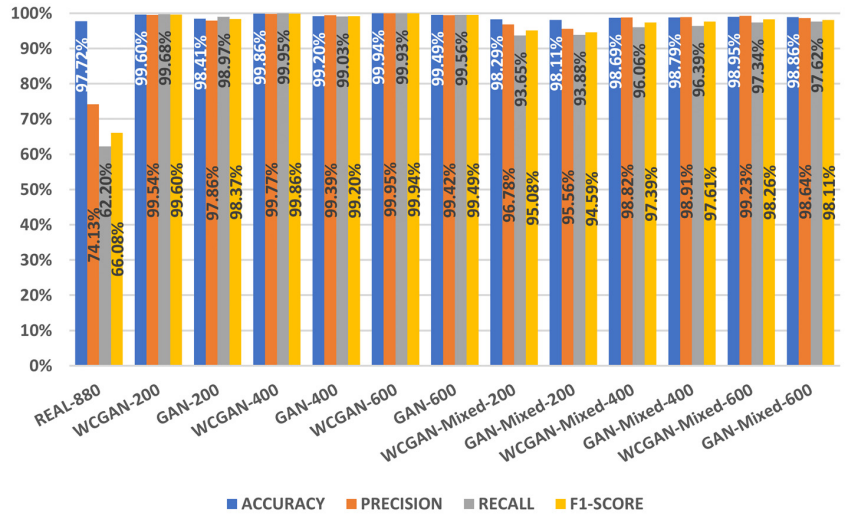

Figure 16. Histograms of the performance of our 13 GB-based classifiers on our 13 test sets.

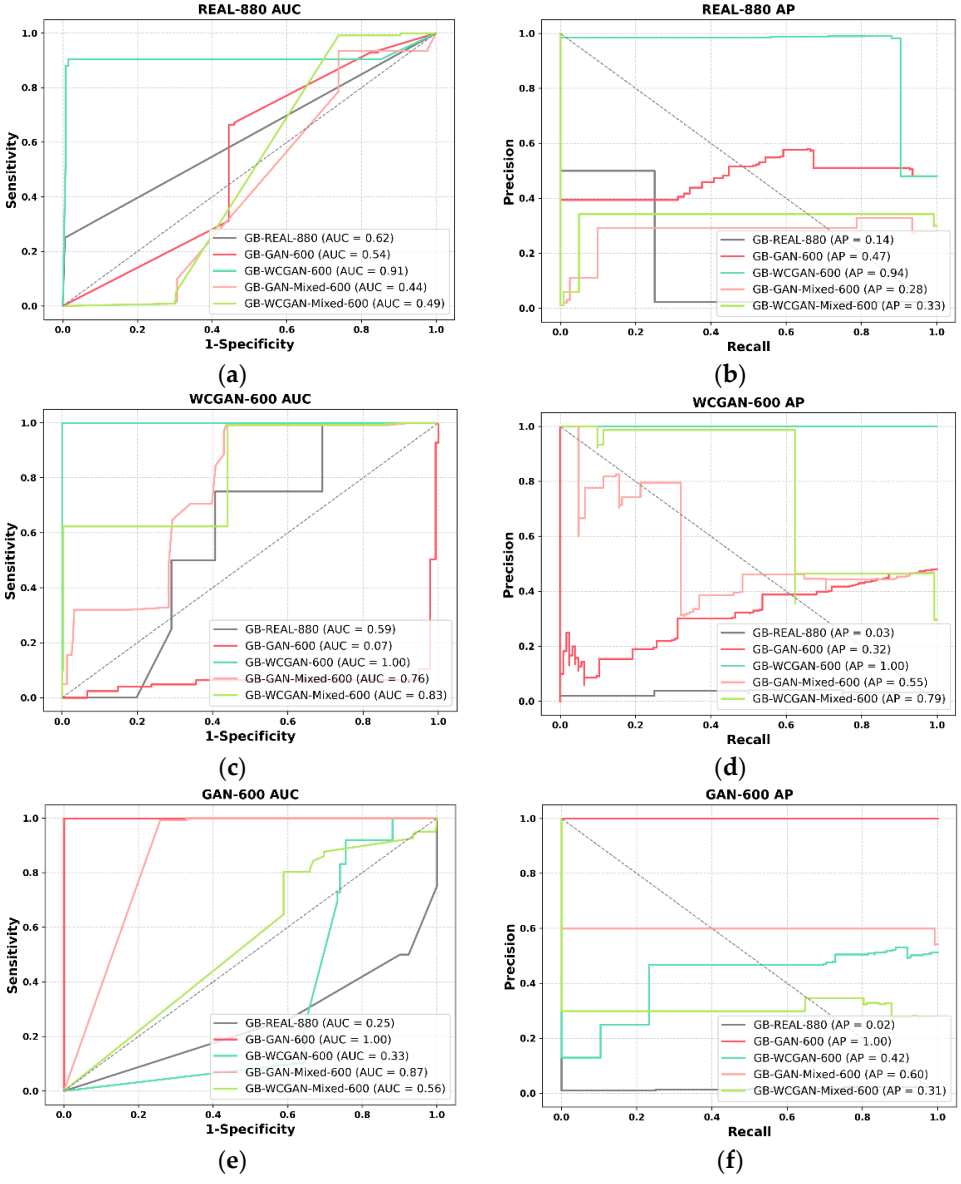

Figure 17. *Cont.*

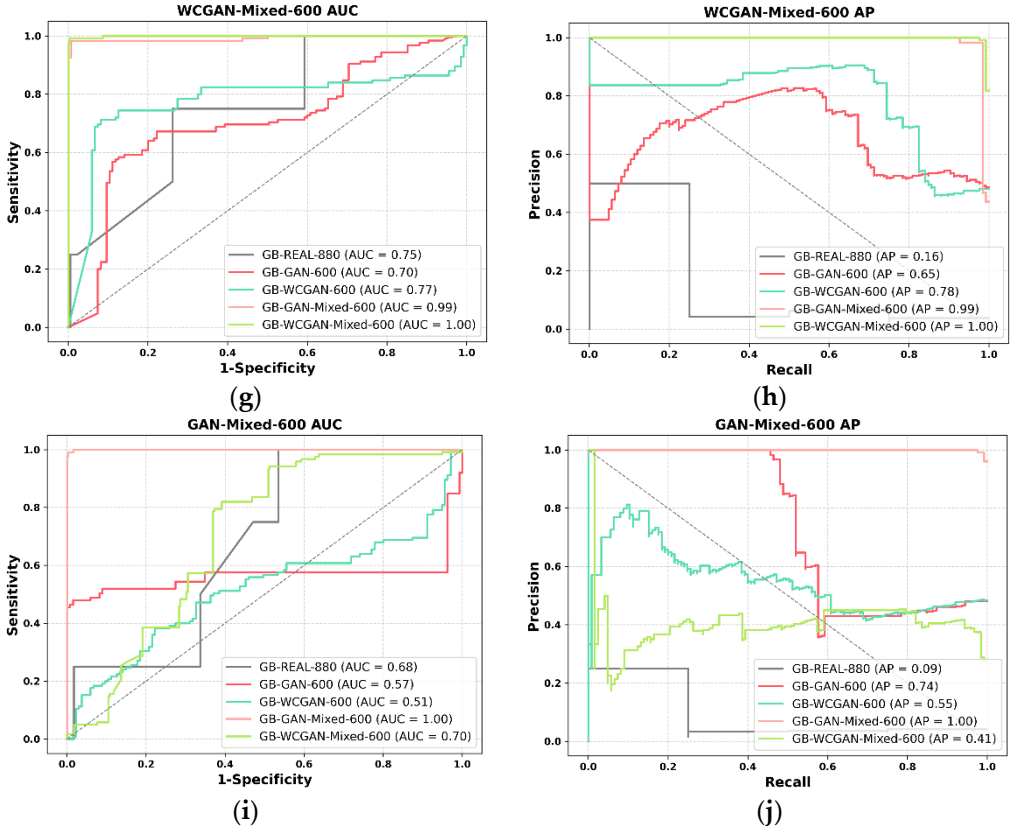

Figure 17. Concept graph presenting ROC and Precision–Recall curves: (**a**) ROC curve for all the GB-based classifiers for the REAL-880 data set; (**b**) Precision–Recall curve for all the GB-based classifiers for the REAL-880 data set; (**c**) ROC curve for all the GB-based classifiers for the WCGAN-600 data set; (**d**) Precision–Recall curve for all the GB-based classifiers for the WCGAN-600 data set; (**e**) ROC curve for all the GB-based classifiers for the GAN-600 data set; (**f**) Precision–Recall curve for all the GB-based classifiers for the GAN-600 data set; (**g**) ROC curve for all the GB-based classifiers for the WCGAN-Mixed-600 data set; (**h**) Precision–Recall curve for all the GB-based classifiers for the WCGAN-Mixed-600 data set; (**i**) ROC curve for all the GB-based classifiers for the GAN-Mixed-600 data set; (**j**) Precision–Recall curve for all the GB-based classifiers for the GAN-Mixed-600 data set.

3.4. Evaluation of Our Best GB-WCGAN-Mixed-600 Classifier with Our GB-REAL-880 Classifier

To evaluate the effectiveness of our new GB-WCGAN-Mixed-600 classifier, we tested our new classifier in the test set of our REAL-880 classifier.

As a result, as shown in Figure 18, our GB-WCGAN-Mixed-600 classifier for the REAL-880 test set performed with 98% Accuracy, 74% Precision, 74.5% Recall and 74.25% F1-score, while the GB-REAL-880 classifier for this set performed with 97.72% Accuracy, 74.13% Precision, 62.20% Recall and 66.08% F1-score. With our new GB-WCGAN-Mixed-600 classifier we had an 8.17% increase in the performance, which is satisfactory due to the lack of a samples test set.

So, from the above our goal of generating new circuit samples based on area, power and time analysis features from the GLN phase is validated, which would enhance the development of a robust ML-based classifier, for the classification of TF and TI circuits. Our new generated data sets, large in size, enhanced the classification of TF and TI circuits. Specifically, throughout this process our first goal was to develop new generated data sets to observe how significantly or not our new data sets could enhance the classification of TF and TI circuits at GLN. Additionally, our next goal was to evaluate if our new data sets could be used as a solution for the problem of a lack of samples, from which the field of countermeasures against HTs suffers. The experimental results prove the achievement of

our goals, as our new WCGAN-Mixed-600 data set managed to develop a more effective classification model for the classification of TF and TI circuits at the GLN phase of ASICs.

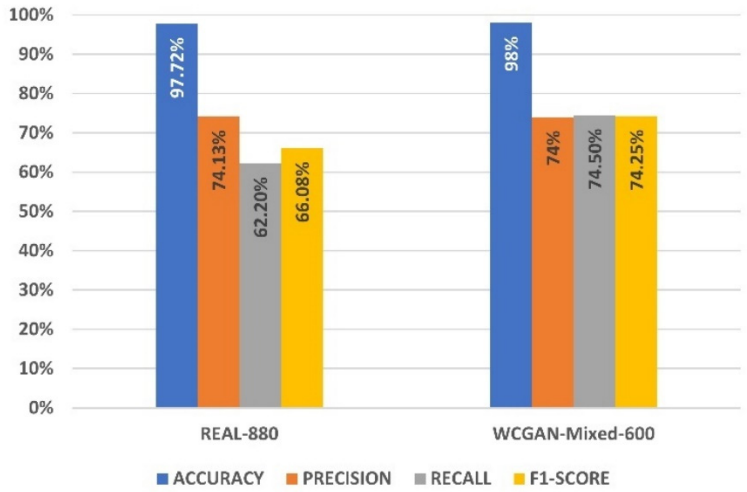

Figure 18. Histograms of the performance of our new best-performing GB-WCGAN-Mixed-600 classifier compared with our GB-REAL-880 classifier on the REAL-880 test set.

4. Conclusions

The HT detection field has been at the forefront of hardware security for the last two decades. As the technological advancements require an ever-increasing complexity level of ICs, the same trend can be observed in HT-based attacks, in their sophistication and elusiveness that prevents detection at pre-silicon stages. However, the pace of advancement has not been the same for the HT detection field, since the development of robust HT detection methods requires abundant data in the form of HT-free and HT-infected circuits. This major obstacle can be attributed to the lack of freely available IC designs, since the majority of ICs are protected by IP rights. Public repositories such as Trust-HUB indeed provide free designs; however, the supported ICs are limited both in terms of absolute numbers and in function/size diversity.

To alleviate the imbalance problem in freely accessible IC design repositories, we propose GAINESIS, a novel approach for generating synthetic HT-free and HT-infected GLN feature vectors in ASICs from a WCGAN-based generative model and high-quality area and power analysis features extracted by the Design Compiler NXT tool. Balanced synthetic data sets of different sizes were generated and utilized to train several ML algorithms that are frequently being applied in the HT detection field. This approach enabled us to evaluate GAINESIS and extract results showing that our method can be effective in generating synthetic feature vectors that can be used for training ML models, which can generalize the original Trust-HUB test set and perform better than the models trained on the original imbalanced data.

Even though GAINESIS is a novel approach that was able to marginally improve (~8% in terms of F1 score) the performance of the original test set, it has the potential to open new research avenues for the HT detection field, as it can also be applied in other pre-silicon IC production phases such as RTL, P&R and GDSII. However, GAINESIS cannot remedy the problem of the lack of numbers and diversity in terms of size and function that is present in Trust-HUB and other freely accessible repositories. To have a better understanding of GAINESIS's ability to provide high-quality synthetic data, we need to assemble a significantly larger and more diverse design set, and more importantly, designs that are derived from real-world applications. For small laboratories, this is a costly and extremely time-consuming effort. Instead, a consortium-level initiative needs to be established where laboratories and companies from all over the world can contribute

to this cause in a crowdsourcing fashion, with the clear purpose of generating large and diverse datasets.

In the future, we will create our own small-in-size circuits, aiming to solve the lack of diversity that is present in Trust-HUB, and through these circuits our GAINESIS tool will be upgraded. In addition, we believe that a more efficient strategy for the detection and mitigation of HT combines different techniques that complement each other. Therefore, we will combine GAINESIS with other run-time and test-time techniques, such as the works in [46–48]. Our GAINESIS tool is available through this link: https://caslab.e-ce.uth.gr/ToolsandDatabases.html (accessed on 10 January 2022).

Author Contributions: Conceptualization, F.C.P. and K.G.L.; methodology, F.C.P. and P.K.; software, K.G.L.; validation, K.G.L. and G.K.G.; formal analysis, K.G.L. and G.K.G.; investigation, K.G.L.; resources, F.C.P.; data curation, K.G.L.; writing—original draft preparation, K.G.L. and G.K.G.; writing—review and editing, F.C.P. and P.K.; visualization, K.G.L.; supervision, F.C.P.; project administration, F.C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available data sets were analyzed in this study. These data can be found here: https://trust-hub.org/#/benchmarks/chip-level-trojan (accessed on 10 January 2022) and here: https://caslab.e-ce.uth.gr/ToolsandDatabases.html (accessed on 10 January 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bhunia, S.; Abramovici, M.; Agrawal, D.; Bradley, P.; Hsiao, M.S.; Plusquellic, J.; Tehranipoor, M. Protection against hardware trojan attacks: Towards a comprehensive solution. *IEEE Des. Test* **2013**, *30*, 6–17. [CrossRef]
2. Samuel, A.L. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* **2000**, *3*, 210–229. [CrossRef]
3. Hinton, G.; LeCun, Y.; Bengio, Y. Deep Learning. *Nature* **2015**, *521*, 436–444.
4. Georgakilas, G.K.; Grioni, A.; Liakos, K.G.; Chalupova, E.; Plessas, F.C.; Alexiou, P. Multi-branch Convolutional Neural Network for Identification of Small Non-coding RNA genomic loci. *Sci. Rep.* **2020**, *10*, 9486. [CrossRef]
5. Pantazi, X.E.; Moshou, D.; Tamouridou, A.A. Automated leaf disease detection in different crop species through image features analysis and One Class Classifiers. *Comput. Electron. Agric.* **2019**, *156*, 94–104. [CrossRef]
6. Liakos, K.G.; Georgakilas, G.K.; Moustakidis, S.; Sklavos, N.; Plessas, F.C. Conventional and Machine Learning Approaches as Countermeasures against Hardware Trojan Attacks. *Microprocess. Microsyst.* **2020**, *79*, 103295. [CrossRef]
7. Hasegawa, K.; Oya, M.; Yanagisawa, M.; Togawa, N. Hardware Trojans classification for gate-level netlists based on machine learning. In Proceedings of the 2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS), Sant Feliu de Guixols, Spain, 4–6 July 2016. [CrossRef]
8. Hasegawa, K.; Yanagisawa, M.; Togawa, N. Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017. [CrossRef]
9. Inoue, T.; Hasegawa, K.; Yanagisawa, M.; Togawa, N. Designing hardware trojans and their detection based on a SVM-based approach. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017. [CrossRef]
10. Liakos, K.G.; Georgakilas, G.K.; Plessas, F.C. Hardware Trojan Classification at Gate-level Netlists based on Area and Power Machine Learning Analysis. In Proceedings of the 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 7–9 July 2021. [CrossRef]
11. Salmani, H.; Tehranipoor, M.; Karri, R. On design vulnerability analysis and trust benchmarks development. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013. [CrossRef]
12. Shakya, B.; He, T.; Salmani, H.; Forte, D.; Bhunia, S.; Tehranipoor, M. Benchmarking of Hardware Trojans and Maliciously Affected Circuits. *J. Hardw. Syst. Secur.* **2017**, *1*, 85–102. [CrossRef]
13. Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive growing of GANs for improved quality, stability, and variation. *arXiv* **2018**, arXiv:1710.10196.
14. Zhang, H.; Xu, T.; Li, H.; Zhang, S.; Wang, X.; Huang, X.; Metaxas, D.N. StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017. [CrossRef]
15. Li, Y.; Liu, S.; Yang, J.; Yang, M.H. Generative face completion. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017. [CrossRef]
16. Zhang, H.; Sindagi, V.; Patel, V.M. Image De-Raining Using a Conditional Generative Adversarial Network. *IEEE Trans. Circuits Syst. Video Technol.* **2020**, *30*, 11. [CrossRef]

17. Design Compiler NXT Tool, Synopsys. Available online: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html (accessed on 10 January 2022).
18. Van Rossum, G.; Drake, F.L. *Python Reference Manual*; Centrum voor Wiskunde en Informatica (CWI): Amsterdam, The Netherlands, May 1995.
19. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, D.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), Savannah, GA, USA, 2–4 December 2016. [CrossRef]
20. Cholle, F. Keras. *J. Chem. Inf. Model.* **2013**, *53*, 1689–1699.
21. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Müller, A.; Nothman, J.; Louppe, G.; et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
22. Chen, T.; Guestrin, C. XGBoost: A scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794. [CrossRef]
23. Kluyver, T.; Ragan-Kelley, B.; Pérez, F.; Granger, B.E.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; et al. *Jupyter Notebooks—A Publishing Format for Reproducible Computational Workflows*; Ebook: Positioning and Power in Academic Publishing: Players, Agents and Agendas; IOS Press BV: Amsterdam, The Netherlands, 2016. [CrossRef]
24. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
25. Cover, T.M. Estimation by the Nearest Neighbor Rule. *IEEE Trans. Inf. Theory* **1968**, *14*, 50–55. [CrossRef]
26. Berkson, J. Application of the Logistic Function to Bio-Assay. *J. Am. Stat. Assoc.* **1944**, *39*, 357–365. [CrossRef]
27. Pal, S.K.; Mitra, S. Multilayer Perceptron, Fuzzy Sets, and Classification. *IEEE Trans. Neural Netw.* **1992**, *3*, 683–697. [CrossRef] [PubMed]
28. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
29. Cortes, C.; Vapnik, V. Support-Vector Networks. *Mach. Learn.* **1995**, *20*, 273–293. [CrossRef]
30. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
31. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets Mehdi. *arXiv* **2014**, arXiv:1411.1784v1.
32. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein generative adversarial networks. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
33. Qin, S.; Jiang, T. Improved Wasserstein conditional generative adversarial network speech enhancement. *Eurasip J. Wirel. Commun. Netw.* **2018**, *2018*, 181. [CrossRef]
34. Oliveira, C.H.M.; Moreira, M.T.; Guazzelli, R.A.; Calazans, N.L.V. ASCEnD-FreePDK45: An open source standard cell library for asynchronous design. In Proceedings of the 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS), Monte Carlo, Monaco, 11–14 December 2016. [CrossRef]
35. Royer, C.W.; O'Neill, M.; Wright, S.J. A Newton-CG algorithm with complexity guarantees for smooth unconstrained optimization. *Math. Program.* **2020**, *180*, 451–488. [CrossRef]
36. Liu, D.C.; Nocedal, J. On the limited memory BFGS method for large scale optimization. *Math. Program.* **1989**, *45*, 503–528. [CrossRef]
37. Fan, R.E.; Chang, K.W.; Hsieh, C.J.; Wang, X.R.; Lin, C.J. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.* **2008**, *9*, 1871–1874. [CrossRef]
38. Schmidt, M.; le Roux, N.; Bach, F. Minimizing finite sums with the stochastic average gradient. *Math. Program.* **2017**, *162*, 83–112. [CrossRef]
39. Defazio, A.; Bach, F.; Lacoste-Julien, S. SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives. In Proceedings of the Advances in Neural Information Processing Systems 27 (NIPS 2014), Montreal, QC, Canada, 8–13 December 2014; pp. 1646–1654.
40. Ruder, S. An Overview Optimization Gradients. *arXiv* **2017**, arXiv:1609.04747.
41. Kingma, D.P.; Ba, J.L. Adam: A method for stochastic optimization. *arXiv* **2015**, arXiv:1412.6980.
42. Kurbiel, T.; Khaleghian, S. Training of Deep Neural Networks Based on Distance Measures Using RMSProp. 2017, pp. 1–6. Available online: http://arxiv.org/abs/1708.01911 (accessed on 10 January 2022).
43. Agarap, A.F. Deep Learning Using Rectified Linear Units (ReLU). No. 1. 2018, pp. 2–8. Available online: http://arxiv.org/abs/1803.08375 (accessed on 10 January 2022).
44. Han, J.; Moraga, C. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation*; Springer: Berlin/Heidelberg, Germany, 1995; pp. 195–201. [CrossRef]
45. Nwankpa, C.; Ijomah, W.; Gachagan, A.; Marshall, S. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. *arxiv* **2018**, arXiv:1811.03378, 1–20.
46. Kitsos, P.; Simos, D.E.; Torres-Jimenez, J.; Voyiatzis, A.G. Exciting FPGA cryptographic Trojans using combinatorial testing. In Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, USA, 2–5 November 2015. [CrossRef]

47. Pyrgas, L.; Kitsos, P. A hybrid FPGA trojan detection technique based-on combinatorial testing and on-chip sensing. In Proceedings of the International Symposium on Applied Reconfigurable Computing, Santorini, Greece, 2–4 May 2018. [CrossRef]
48. Fournaris, A.P.; Pyrgas, L.; Kitsos, P. An efficient multi-parameter approach for FPGA hardware Trojan detection. *Microprocess. Microsyst.* **2019**, *71*, 102863. [CrossRef]