ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ευφυές Υπολογιστικό Σύστημα για την Ανίχνευση Ελαττωμάτων και Ανωμαλιών στην Ηλεκτρολογία

Διατριβή η οποία υποβλήθηκε για τη μερική εκπλήρωση των υποχρεώσεων απόκτησης του Διδακτορικού Διπλώματος

Κωνσταντίνος Λιάκος

Μάιος 2022

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intelligent Computational System for Defects and Anomalies Detection in Electrical Engineering

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Konstantinos Liakos

May 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ευφυές Υπολογιστικό Σύστημα για την Ανίχνευση Ελαττωμάτων και Ανωμαλιών στην Ηλεκτρολογία

Διδακτορική Διατριβή

Κωνσταντίνος Λιάκος

Συμβουλευτική Επιτροπή

Πλέσσας Φώτιος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας. (Επιβλέπων)

Κίτσος Παρασκευάς, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πελοποννήσου

Leporati Francesco, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Παβία

Επταμελής εξεταστική επιτροπή

Πλέσσας Φώτιος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας. (Επιβλέπων)

Κίτσος Παρασκευάς, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πελοποννήσου

Leporati Francesco, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Παβία

Σταμούλης Γεώργιος, Καθηγητής, Πανεπιστήμιο Θεσσαλίας

Σωτηρίου Χρήστος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας

Σκλάβος Νικόλας, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Πατρών

Ποταμιάνος Γεράσιμος, Αναπληρωτής Καθηγητής, Πανεπιστήμιο Θεσσαλίας

Μάιος 2022

ΥΠΕΥΘΥΝΗ ΔΗΛΩΣΗ ΠΕΡΙ ΑΚΑΔΗΜΑΪΚΗΣ ΔΕΟΝΤΟΛΟΓΙΑΣ ΚΑΙ ΠΝΕΥΜΑΤΙΚΩΝ ΔΙΚΑΙΩΜΑΤΩΝ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα διδακτορική διατριβή, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της διατριβής, αποτελούν αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλουν οποιασδήποτε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχουν έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Δηλώνω επίσης ότι τα αποτελέσματα της εργασίας δεν έχουν χρησιμοποιηθεί για την απόκτηση άλλου πτυχίου. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο Δηλών

Κωνσταντίνος Λιάκος

UNIVERSITY OF THESSALY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intelligent Computational System for Defects and Anomalies Detection in Electrical Engineering

PH.D. Dissertation

Konstantinos Liakos

Advisory Committee

Plessas Fotios, Associate Professor, University of Thessaly

Kitsos Paris, Associate Professor, University of Peloponnese

Leporati Francesco, Associate Professor, University of Pavia

Examination Committee

Plessas Fotios, Associate Professor, University of Thessaly

Kitsos Paris, Associate Professor, University of Peloponnese

Leporati Francesco, Associate Professor, University of Pavia

Stamoulis George, Professor, University of Thessaly

Sotiriou Christos, Associate Professor, University of Thessaly

Sklavos Nikolas, Associate Professor, University of Patra

Potamianos Gerasimos, Associate Professor, University of Thessaly

May 2022

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this Ph.D. dissertation, as well as the electronic files and source codes developed or modified in the course of this dissertation, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this dissertation or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Konstantinos Liakos

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διδακτορικής διατριβής, θα ήθελα να ευχαριστήσω θερμά τα άτομα που συνέβαλαν με τη συνεισφορά τους, για την ολοκλήρωση της. Συγκεκριμένα, θέλω να ευχαριστήσω τον αγαπημένο μου εποπτεύων καθηγητή Πλέσσα Φώτιο και τους καθηγητές Κίτσο Παρασκευά και Leporati Francesco, καθώς και τον αγαπημένο μου φίλο δόκτωρ Γεωργακίλα Γεώργιο, για την βοήθεια και την υποστήριξη τους καθ' όλη την διάρκεια του διδακτορικού μου. Τέλος, θα ήθελα να ευχαριστήσω ολόψυχα την σύζυγο μου Μαίρη, τους πολυαγαπημένους μου γονείς, τον πατέρα μου Γεώργιο και την μητέρα μου Μαρία, τα αδέρφια μου Ειρήνη και Νικόλα και τον ανιψιό μου Χρήστο, οι οποίοι είναι η χαρά της ζωής μου και η δύναμη μου όλα τα χρόνια και δείχνουν την αμέριστη συμπαράστασή τους ως προς το πρόσωπό μου, τόσο στο κομμάτι των σπουδών όσο και της ζωής και βρίσκονται πάντα δίπλα μου.

Διδακτορική Διατριβή

Ευφυές Υπολογιστικό Σύστημα για την Ανίχνευση Ελαττωμάτων και Ανωμαλιών στην Ηλεκτρολογία

Κωνσταντίνος Λιάκος

Περίληψη

Στις μέρες μας υπάρχει ανάγκη για ακόμη πιο εξελιγμένα κυκλώματα. Οι εταιρείες σχεδιασμού, προκειμένου να μειώσουν το λειτουργικό κόστος και να διευκολύνουν τη μαζική παραγωγή ολοκληρωμένων κυκλωμάτων, αναθέτουν την κατασκευή τους σε τρίτους. Η διαδικασία αυτή αυξάνει τον κίνδυνο επιθέσεων εισβολής με τη μορφή ιών υλικού, γνωστών και ως ιών δουρείου ίππου συσκευών. Οι ιοί αυτοί αποτελούν ένα σημαντικό πρόβλημα που έχει τη δυνατότητα να εξελιχθεί σε επιδημία τα επόμενα χρόνια, αποτελώντας σημαντική απειλή τόσο από τεχνολογική όσο και από κοινωνική άποψη.

Η πλειονότητα των μελετών αφορά την ανάπτυξη αντιμέτρων κατά των ιών δουρείου ίππου συσκευών, για κυκλώματα συστοιχίας προγραμματιζόμενων πυλών πεδίου και συγκεκριμένα για το στάδιο μετά το πυρίτιο. Επίσης, υπάρχουν περιορισμένες πληροφορίες και δημοσιευμένες μελέτες για τα ολοκληρωμένα κυκλώματα ειδικής εφαρμογής και συγκεκριμένα για το στάδιο πριν από το πυρίτιο. Τα ολοκληρωμένα κυκλώματα ειδικής εφαρμογής αποτελούν πρόκληση λόγω της ποικιλίας των φάσεων σχεδιασμού που έχουν και ιδίως στο στάδιο πριν από το πυρίτιο, καθώς και λόγω της ανάγκης επαγγελματικών εργαλείων για το σχεδιασμό κάθε φάσης.

Σε αυτή τη διατριβή μελετήσαμε διάφορες φάσεις για τη διαδικασία σχεδιασμού σε ολοκληρωμένα κυκλώματα ειδικής εφαρμογής και διαπιστώσαμε ότι υπάρχει γενική έλλειψη σε δεδομένα από δημόσια-ελεύθερα κυκλώματα καθώς ότι υπάρχει επίσης μεγάλο πρόβλημα ανισορροπίας μεταξύ μη μολυσμένων και μολυσμένων κυκλωμάτων. Χρησιμοποιήσαμε και σχεδιάσαμε όλα τα περιορισμένα κυκλώματα αναφοράς, για τη φάση επιπέδου πύλης των ολοκληρωμένων κυκλωμάτων ειδικής εφαρμογής, με ένα επαγγελματικό εργαλείο και εξήγαμε χαρακτηριστικά ανάλυσης εμβαδού, ισχύος και χρόνου. Αναπτύξαμε τα δικά μας μοντέλα ταξινόμησης μηχανικής μάθησης, με βάση αυτά τα περιορισμένα δεδομένα και παρατηρήσαμε ότι η έλλειψη δειγμάτων οδηγεί στην

ανάπτυξη ανισόρροπων και μη ισχυρών μελετών ταξινόμησης που βασίζονται σε μηχανική μάθηση, για την αντιμετώπιση των ιών δουρείου ίππου συσκευών. Επιλύσαμε το πρόβλημα των περιορισμένων δεδομένων με την ανάπτυξη των δικών μας μοντέλων βαθιάς μάθησης - γενετικών αντιθετικών δικτύων, τα οποία ήταν σε θέση να συνθέσουν νέα παραγόμενα δεδομένα με βάση τα πραγματικά περιορισμένα δεδομένα μας. Τα γενετικά αντιθετικά δίκτυα είναι νέοι αλγόριθμοι βαθιάς μάθησης που χρησιμοποιούνται στον τομέα της υπολογιστικής όρασης, για τη δημιουργία τεχνητών εικόνων. Ήταν η πρώτη φορά που χρησιμοποιήθηκαν γενετικά αντιθετικά δίκτυα σε αυτό το ερευνητικό πεδίο. Έτσι, με βάση τα νέα παραγόμενα δεδομένα μας αναπτύξαμε έναν ισχυρό ταξινομητή βασισμένο σε μηχανική μάθησης, ως αντίμετρο κατά των ιών δουρείου ίππου συσκευών για την φάση επιπέδου πυλών και τον συγκρίναμε με υπάρχουσες μεθόδους για αυτή την φάση. Τέλος, μετατρέψαμε το παραγωγικό μας μοντέλο σε ένα ελεύθερο εργαλείο προκειμένου να χρησιμοποιηθεί ως λύση για την αντιμετώπιση του περιορισμένου αριθμού δεδομένων.

Λέξεις-κλειδιά:

Ασφάλεια υλικού, ολοκληρωμένα κυκλώματα, ολοκληρωμένα κυκλώματα ειδικής εφαρμογής, ιοί δουρείου ίππου συσκευών, αντίμετρα, τεχνητή νοημοσύνη, μηχανική μάθηση, μάθηση σε βάθος, παραγωγική μάθηση, παραγωγικά αντιφατικά δίκτυα, προ πυριτίου στάδιο, φάση επιπέδου πύλης

Ph.D. Dissertation

Intelligent Computational System for Defects and Anomalies Detection in Electrical Engineering

Konstantinos Liakos

Abstract

In our days there is a need for even more and more sophisticated circuits. The design companies to reduce the operating costs and facilitate mass production of integrated circuits, outsource their 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) viruses. HTs viruses are a critical problem that has the potential to become an outbreak in the coming years, presenting a significant threat both technologically and socially.

The majority of the studies are concerned with the development of countermeasures against HTs for Field-Programmable Gate Array (FPGA) circuits at the post-silicon stage. Also, there is limited information and published studies for the Application-Specific Integrated Circuits (ASICs) and specifically for the pre-silicon stage. ASICs are challenging due to the variety of design phases especially on the pre-silicon stage and for the need for professional tools for the design of each phase.

In this thesis, we studied several phases for the design process on ASICs and we found that there is a general lack of free benchmark circuits and also there is a high imbalance problem between uninfected and infected benchmark circuits. We used and designed all the limited benchmark circuits for the Gate-Level Netlist (GLN) phase of ASICs with a professional tool and extracted area, power and time analysis features. We developed our Machine Learning (ML) classification models based on this limited data and we observed that the lack of samples leads to the development of imbalanced and no robust ML-based classification approaches against HTs viruses. We solved the problem of the limited data with the development of our Deep Learning (DL) - Generative Adversarial Networks (GANs) models which were able to synthesize new generated data based on our real limited data. GANs are novel DL algorithms that are used in the computer vision field for generating artificial images and it was the first time that GANs were used in this research field. Based on our

new generated data we developed a robust ML-based classifier as a countermeasure against HTs at the GLN phase and compared it with existing methods. Finally, we turned our generative model into a free tool to be used as a solution for dealing with the limited number of data.

Keywords:

Hardware security, integrated circuits, application-specific integrated circuits, hardware trojan viruses, countermeasures, artificial intelligence, machine learning, deep learning, generative learning, generative adversarial networks, pre-silicon stage, gate-level netlist phase

Πίνακας περιεχομένων

Κατάλογος εικόνων

Κατάλογος πινάκων

Συντομογραφίες

Application-Specific Integrated Circuit ASIC
Area Under Curve AUC
Artificial Intelligence AI
Artificial Neural Networks ANNs
Average Precision AP
Bayesian Models BM
Computer Process Unit CPU
Computer vision CV
Conditional Generative Adversarial Networks CGANs
Convolutional Neural Networks CNNs
Coordinate Descent CD
Decision Trees DT
Deep Learning DL
Deep Neural Networks DNNs
Dimensionality Reduction DR
Electronic Design Automation EDA
Ensemble Learning EL
False NegativeFN
False Positive FP
Field-Programmable Gate Arrays FPGA
Gate Level Netlist GLN
Generative Adversarial Networks GANs
Generative Artificial Intelligence NEtlists SynthesIS GAINESIS
Generative Learning GL
Gradient Boosting GB
Graphic Database System II GDSII
Graphic Process Unit GPU
hArdware Trojan Learning AnalysiS ATLAS
Hardware Trojans HTs
Instance Based IB
Integrated Circuits ICs
Intellectual Property IP
Internet of Things IoT
K-Nearest Neighbors KNN
Limited-Memory Broyden–Fletcher–Goldfarb–Shanno LM-BFGS
Logic Testing LT
Logistic Regression LR
Machine Learning ML
Multilayer Perceptron MLP
Natural Language Processing NLP
Placement & Routing P&R
Prevention-Facilitation PF
Random Forest RF
Receiver Operating Characteristic ROC
Rectified Linear Unit ReLU

Register Transfer Level RTL
Root Mean Square propagation RMSprop
Runtime Monitoring RM
Side Channel Analysis SCA
Speech Recognition SR
Stochastic Average Gradient SAG
Stochastic Gradient Descent SGD
Support Vector Machine SVM
Tangent Tanh
Trojan-Free TF
Trojan-Infected TI
True Negative Rate TNR
True Negative TN
True Positive Rate TPR
True Positive TP
Wasserstein Conditional Generative Adversarial Network WCGAN
Wasserstein Generative Adversarial Networks WGAN
Xtreme Gradient Boosting XGB

Chapter 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 has the potential to become an outbreak in the coming years, presenting a significant threat both technologically and socially. The majority of the studies are concerned for the development of countermeasures against HTs for Field-Programmable Gate Array (FPGA) circuits at post-silicon stage. There is a limited information and published studies for the ASICs and specifically for the pre-silicon stage [1–25]. ASICs are challenging due to the variety of design phases especially on the pre-silicon stage and for the need of professional tools for the design of each phase.

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 [16]. Therefore, developing well-designed and efficient HT countermeasures is crucial. The HT structure consists of an activation mechanism (trigger) and an effect (payload). HTs remain totally silent and via rare events or signals their triggers are activated [16], 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.

The question that quickly comes to mind is, who gains from the insertion of HTs into ICs? A competitor, for example, might put an infected circuit into another company's IC to discredit it, diminish its market share, consumer confidence, and earnings. Another HT use case involves the sabotage of military equipment and infrastructure between countries through HT cyber warfare [17].

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), Gate Level Netlist (GLN), Placement & 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. 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.

Machine Learning (ML) [18] and Deep Learning (DL) [19] 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. [20][21]. In the hardware security field, a plethora of ML-based approaches for HT detection has been introduced in the last six years [22][23]. 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 [24][25][26][12].

Most ML-based studies in the field of HTs utilize the public Trust-HUB [28][29] 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 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 Intellectual Property (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.

1.1 Motivation and Structure of the Dissertation

The main topic of this research is 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: Generative Artificial Intelligence NEtlists SynthesIS. GAINESIS utilizes a Wasserstein Conditional Generative Adversarial Network (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 [30], textual descriptions of birds and flowers [31], reconstructing damaged photographs of human faces [32], removing rain and snow from photographs [33] and many other functions. For the development of GAINESIS, the Design Compiler NXT tool 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. Also, 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. The best performed GB-classifier was picked as our main classifier with the name ATLAS: hArdware Trojan Learning AnalysiS and compared to existing methods at the same unknown benchmark.

The remaining part of this dissertation is organized as follows: a detailed description of the HTs is given in Chapter 2. Specifically, are mentioned in detail HTs structure, models, attacks and taxonomy.

An overview of AI is presented in Chapter 3. First, we present the terms about AI, ML and DL. Next, we present the tasks of learning and the differences and then we present the most significant types of learning models.

Countermeasures against HTs are presented in Chapter 4. Specifically, we present a historical throwback of countermeasures. Then we present the categorization of the studies to journal and conference approaches. Also, we present the distribution of the most contributing journal and conference studies and we show the studies trend through the years. Lastly, we present in detail the categorization of countermeasures approaches against HTs through three main categories and six sub-categories. Specifically, we mention the function of each represented sub-category and category with tables and figures and we present in aggregate the function, benchmark and features for the approaches for each category.

Chapter 5 presents the methodology of our GAINESIS approach. First, we present our scheme of GAINESIS methodology. Next, we mention our data set and features development. Then, we present our ML-based classifiers development for the classification of HT-free and HT-infected circuits and their evaluation. Specifically, seven different ML algorithms were used and compared for the development of our main classifier. Next, we present the development and evaluation of our GAINESIS approach. Lastly, we present our new generated and mixed data sets, as well as the development of our new generated and mixed based classifiers.

Chapter 6 presents the results of our classifiers for new generated and mixed classifiers compared with our initial classifier. Finally, we present the comparison of our ATLAS classifier with existing state-of-the-arts methods.

Conclusions and future work are presented in Chapter 7.

Chapter 2 Background

2.1 Integrated Circuits Supply Chain

To have a thorough grasp of the topic of HTs, the difficulty of preventing their contagious nature, and the challenges of identifying them while ensuring the smooth operation of ICs, we must first have a strong understanding of the modern circuit production chain and especially the production chain of the ASICs. ASICs production chain consists of two stages, pre- and post-silicon stages. The pre-silicon stage is the circuit design period and consists of steps: RTL, GLN and P&R. And the post-silicon stage is the fabrication period of the circuit and consists of the Side Channel Analysis (SCA) phase.

Specifically, at RTL phase describes the specifications that the circuit will have through the usage of a Hardware Design Language (HDL) like Verilog or VHDL. When IC design and integration are completed at RTL, the design must be synthesized to a GLN. GLN is characterized as the logic synthesis phase and RTL is translated to GLN. The logic synthesis phase is done via professional Electronic Design Automation (EDA) tools like Cadence Genus Synthesis Solution, Synopses Design Compiler NXT etc.). These tools provide area, power and timing analysis of the circuit. The last phase is the P&R and is known as the physical design phase where the layout level is created via the GLN and is produced the final GDSII of the circuit.

So, HT attacks are divided into four general groups for the pre-silicon stage (Figure 2.1), i.e., RTL, GLN, P&R and GDSII as well as Fabrication and Testing/Assembly for the post-silicon stage. Depending on the targeted phase, the attacker might obtain full access to source code and design files, or compromise computer aided design tools and scripts to output a modified IC representation without altering the source code. On the other hand, fabrication attacks take place after tape-out and can add or remove components via reverse engineering, layout geometry modification or IC metering (Figure 2.1).

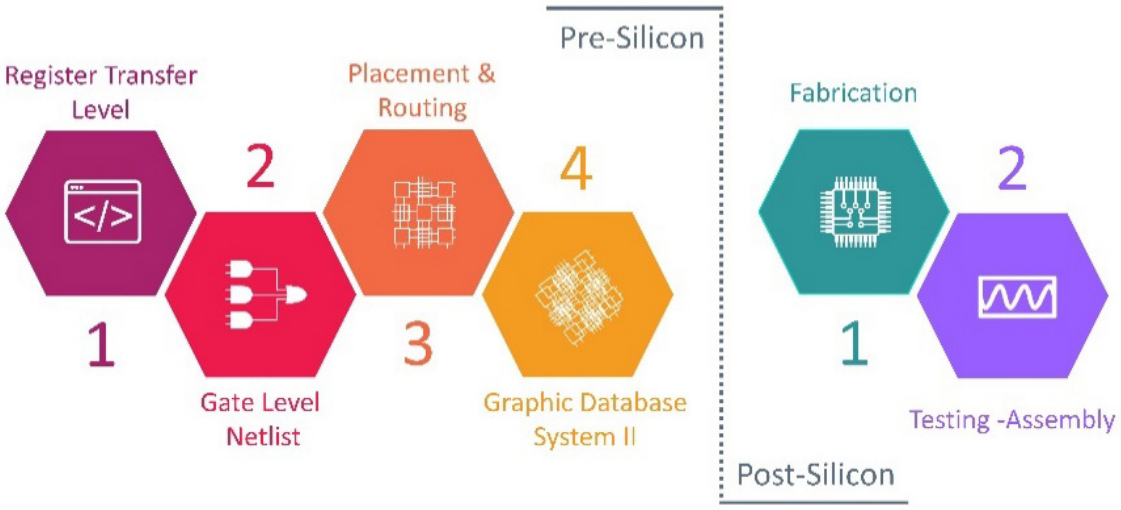


Figure 2.1 IC supply chain and HTs insertion in pre- and post-silicon stages

2.2 Hardware Trojan Structure

The typical structure of an HT consists of two mechanisms, triggers and payloads (Figure 2. 2). Triggers are related to rare signals or events [34] and payloads with the activation of malicious functions. An HT aims to remain stealthy - to be undetectable during design simulation or testing and to be activated under rear conditions. So, an HT "wakes up" when the rare signal or event appears and via the payload mechanism attacks the IC.

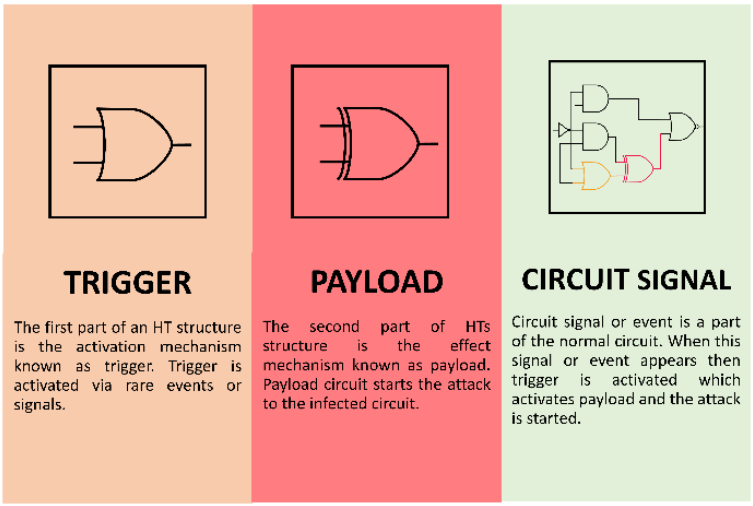


Figure 2.2 Hardware Trojan structure

2.3 Hardware Trojan Models

As mentioned HTs are designed to be undetectable, their structure is consisted of a trigger and a payload mechanism and can be implemented in all pre- and post-silicon phases of the

ICs production chain. Another characteristic of HTs is their logic models. Logic models are associated with the trigger mechanism and especially how the rare signal or event will activate the trigger mechanism. HTs are designed to have two logic models, a combinational or a sequential [34]. In combinational logic models the trigger mechanism is activated from a set of simultaneous rare signals or events (Figure 2.3A) and in sequential logic models from a series of rare events or signals (Figure 2.3B).

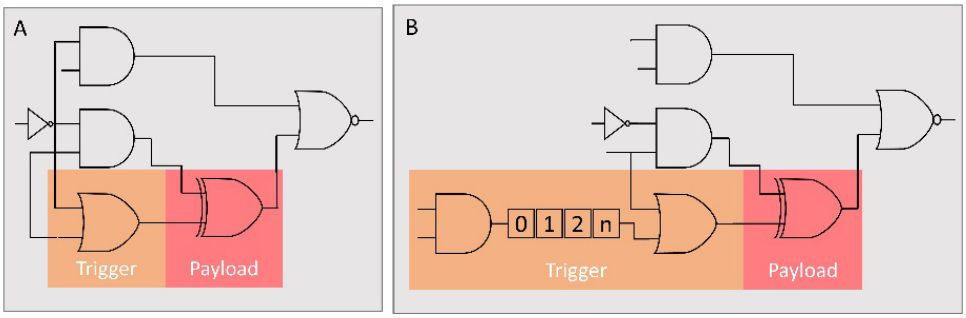


Figure 2.3 Concept graph presenting (A) combinational and (B) sequential model logics

2.4 Hardware Trojan Attacks

The aim of HTs is to affect the normal functioning of the infected circuit. Thus, the HTs attacks can be divided into two types of attacks: those aimed at destroying the device known as general purpose processors attacks and those aimed at leaking sensitive information, known as cryptographic engine attacks. Cryptographic engine attacks aim at the crypto engine of the infected circuit through various attack mechanisms and leak encrypted information. General purpose processors attacks aim at the lower levels of the processor, kernel, memory and secret keys and degrade the system, even down to its total destruction. For example, these types of HTs can be activated under rare signals or events and disable the secure boot mechanism of the infected circuit [35][36].

2.5 Hardware Trojan Taxonomy

There is no formal taxonomy for HTs. Each study has its taxonomy structure. Tehranipoor et al. [37] presented a taxonomy of HTs based on three main characteristics of HTs, physical, activation and action. As physical characteristics are considered the type, size or structure of an HT. Activation characteristics are divided into external and internal activation mechanisms of an HT and action characteristics are considered the types of HT attacks to the infected circuit. Karri et al. [38] proposed a taxonomy model for HTs, based on five

characteristics: insertion phase, abstraction level, activation mechanism, effect and localization. While Bhunia et al. [34], proposed a taxonomy model based on trigger and payload mechanisms.

2.6 Challenges Against Hardware Trojan

Dealing with HTs has become one of the most important problems in the science of hardware security. Every year new studies are developed to address them. The main reason for the difficulty in dealing with HTs is main a large number of different cases of HT infections. HTs can be inserted at any stage and phase of ICs development, can attack at any unit of the ICs, processors, memory units, etc., Also, HTs can affect the ICs via a variety of attacks and can have different physical layouts. In addition, the stealthy nature of HTs and their ability to activate under rare conditions combined with the fact that the more complex a circuit is, the more difficult it is to deal with.

Chapter 3 An Overview on Artificial Intelligence

3.1 Introduction

Every year more and more people refer to terms like AI, ML and DL. This happens because a technology trend is the development and use of AI-based technologies on a professional or personal level. As a result, the meaning of these terms has been lost. So, it is important to understand that all these terms are part of the AI scientific field.

In this chapter a detailed reference is made to the science of AI. Specifically, this chapter of the thesis is has presented differences between the AI, ML and DL terms. Also, are presented with details the learning tasks of AI like, supervised and unsupervised learning. Furthermore, a plethora of learning models and algorithms are discussed exhaustively. The aim of this chapter is for the readers to be able to distinguish the differences between the AI, ML and DL, as well as to comprehend how each learning model works and when their algorithms are applied.

3.2 Artificial Intelligence Term

The term the modern AI first was introduced in 1956 by John McCarthy through an academic conference. McCarthy defined AI as the science of making intelligent machines. So, AI can be defined as the scientific field that aims to teach machines to think without the need for human intervention. AI consists of a broad area of computer science and can be categorized into three main categories, AI-narrow, AI-general and AI-super. AI-narrow is goal-oriented and has been programmed to complete a single task. AI-general allows machines to learn and apply their intelligence to solve any problem by mimicking human intellect and/or behaviors and in AI-super machines are capable of outperforming even the best humans in terms of intelligence.

3.3 Machine Learning Term

ML term was introduced in 1959 by Samuel et al. [18] and it was defined as the scientific field that allows machines to learn without being strictly programmed. Specifically, ML consists of a subset of AI that uses statistical learning algorithms for the development of smart systems. Without being explicitly programmed, ML-based systems can learn and

improve on their own. The ML algorithms can be categorized into three main categories, supervised, unsupervised and semi-supervised learning.

3.4 Deep Learning Term

DL is a subset of ML techniques utilizing multiple layers of training with more reliable performance and fastest speed. The DL technique was inspired by the way a human brain analyzes information. DL-based systems consist of interrelated layers for the classification or prediction of information. In Figure 3.1 is presented in brief the differences between AI, ML and DL.

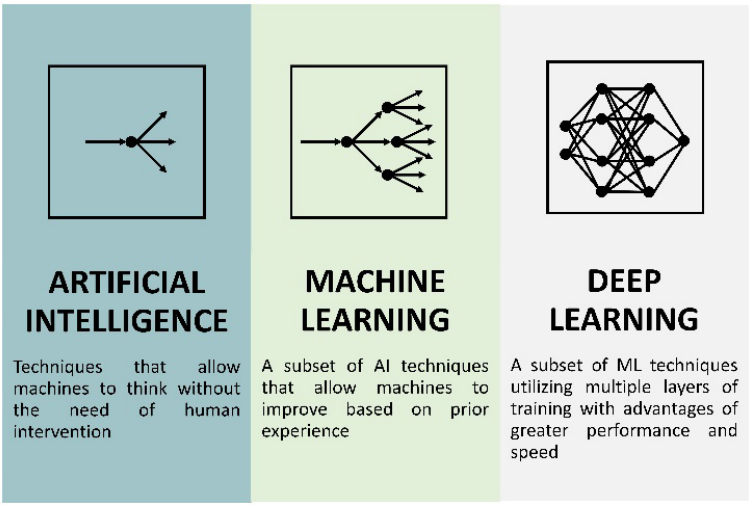


Figure 3.1 Artificial intelligence vs machine learning vs deep learning

3.5 Tasks of Learning

AI, ML or DL algorithms can be categorized into three categories of learning tasks, supervised, unsupervised and semi-supervised learning. The main difference is that supervised learning uses labeled data to help in prediction, while unsupervised does not. Semi-supervised learning uses data mixed with labeled and unlabeled examples. However, there are some distinctions between the three techniques, as well as key areas where one surpasses the others. In this section are presented the differences between the three learning tasks.

3.5.1 Supervised Learning

Supervised learning uses data sets with labeled samples as inputs and outputs for the development of an ML or DL-based model. Supervised learning can be used as a solution

for two categories of problems, classification or regression. In the classification problems a labeled data set is split into sets, the training and test set for the development of a model. The aim is the model to be able to classify with high performance the samples of the test set. For example, a classic supervised classification learning problem is the classification of original from spam emails. Furthermore, in the regression problems aim of the model is through a labeled data set to understand the relationship between dependent and independent variables of the data set. Regression models are useful for predicting numerical values based on various data samples, such as sales revenue estimates for a certain business. In Figure 3.2 is presented a typical figure of supervised learning.

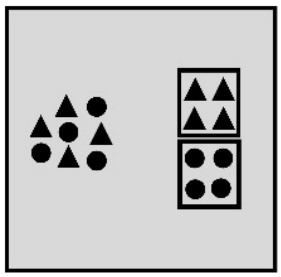


Figure 3.2 Supervised learning

3.5.2 Unsupervised Learning

Unsupervised learning uses data sets with unlabeled samples as inputs and outputs for the development of an ML or DL-based model. In unsupervised learning-based models from the data set it derives patterns between the features and when the model analyzes new data, it can classify the new samples into a class, based on the already learned feature patterns. Unsupervised learning can be used as a solution for clustering or dimensionality reduction problems. In the clustering problems aim of the model is via an unlabeled data set to group the data set. In dimensionality reduction problems aim of the model is to convert the higher dimensions data set into lesser dimensions without losing information, to reduce the poor performance which is produced from the data sets with a large number of features. In Figure 3.3 is presented a typical figure of unsupervised learning.

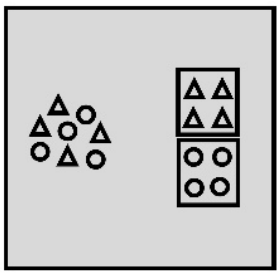


Figure 3.3 Unsupervised learning

3.5.3 Semi-supervised Learning

Semi-supervised learning uses data sets with mixed samples like, labeled and unlabeled samples as inputs and outputs for the development of an ML or DL-based model. There is a desirable prediction problem, but the model must learn the structures to arrange the data and produce predictions. Classification and regression are two common semi-supervised problems. Unsupervised and semi-supervised learning may be more tempting options because relying on domain expertise to label data accurately for supervised learning can be time-consuming and costly. In Figure 3.4 is presented a typical figure of semi-unsupervised learning.

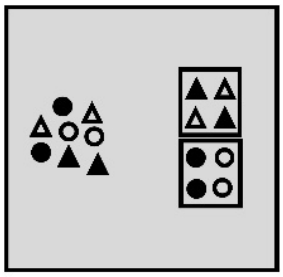


Figure 3.4 Semi-supervised learning

3.6 Types of Learning Models

3.6.1 Artificial Neural Networks Models

Artificial neural networks (ANNs) are inspired by the functionality of the human brain. ANNs emulate complicated tasks like cognition, learning, decision making and pattern generation [39]. The human brain is made up of billions of neurons that communicate with one another and process any information that is sent to them. Based on the same philosophy, an ANN is a simplified model of the structure of a biological neural network, which is made up of interconnected processing units that are organized in a specific topology. Specifically, ANNs consist of three categories of layers, input, hidden and output layers. Input layers fed the data set into the system. Hidden layers produce the learning of the model and the

decision/prediction is given from the output layer. ANNs are supervised models that are commonly used to solve regression and classification problems. The most common ANNs-based algorithms are perceptron [40], multi-layer perceptron [41], back-propagation [42], resilient back-propagation [43] and counter propagation algorithms [44]. Also, other common ANNs algorithms are radial basis function networks [45], Kohonen networks [46], Hopfield networks [47], generalized regression networks [48], autoencoder [49], adaptive-neuro fuzzy inference systems [50], extreme learning machines [51] and self-adaptive evolutionary extreme learning machines [52]. In Figure 3.5 is presented a typical structure of an ANN model.

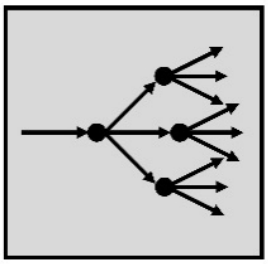


Figure 3.5 Artificial neural networks model

3.6.2 Bayesian Models

Bayesian models (BM) are a type of probabilistic graphical model in which the analysis is carried out using Bayesian inference. This model belongs to the domain of supervised learning and can be used to solve classification or regression problems. Some of the most common BM-based algorithms are Bayesian network [53], bayesian belief network [54], naive Bayes [55], multinomial naive Bayes [56] and Gaussian naive Bayes [57]. In Figure 3.6 is presented a typical figure of a Bayesian model.

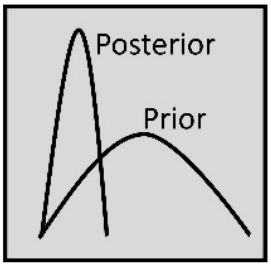


Figure 3.6 Bayesian model

3.6.3 Clustering Models

As mentioned, clustering-based models [58] are typical applications of unsupervised learning models. These types of models are used to find natural groupings of data, known as clusters. Common clustering algorithms are the k-means [59], hierarchical clustering [60] and the expectation maximisation algorithm [61]. In Figure 3.7 is presented a typical structure of a cluster-based model.

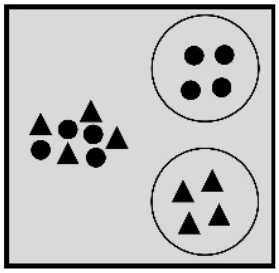


Figure 3.7 Clustering model

3.6.4 Computer Vision Models

Computer vision (CV) models aim to understand information from digital images or videos. CV-based models are concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding. Some of the most common algorithms are, HRNet-OCR [62], FixEfficientNet [63] and EfficientDet [64]. In Figure 3.8 is presented a typical structure of a CV model.

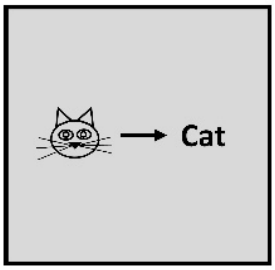


Figure 3.8 Computer vision model

3.6.5 Decision Trees Models

Decision trees (DT) consist of classification or regression models based on a tree-like architecture [65]. In DT-based models, the data set is progressively grouped into smaller homogeneous subsets known as sub-populations, while an associated tree graph is produced

simultaneously. Each internal node of the tree structure reflects a separate pairwise comparison on a given feature, and each branch indicates the outcome of this comparison. Following the path from the root to the leaf, leaf nodes represent the final prediction or decision of the process. Common DT-based algorithms are classification and regression trees [66], chi-square automatic interaction detector [67], and the iterative dichotomiser [68]. In Figure 3.9 is presented a typical structure of a DT model.

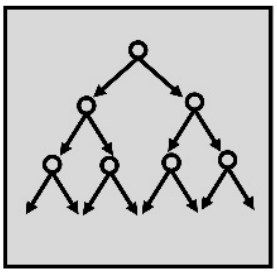


Figure 3.9 Decision trees model

3.6.6 Deep Neural Networks Models

Deep neural networks (DNNs) [69] consist of a modern version of ANNs. DL-based models consist of the new era of AI while more and more models are developed based on them. As the ANNs, the DL-based models consist of three categories of layers, input, multiple hidden and output layers. The significant difference between ANNs is the usage of multiple processing layers which can learn complex data representations via multiple levels of abstraction. Furthermore, one more advantage of DL-based models is that the feature extraction can be performed by the model itself. These models can be used for supervised, unsupervised and semi-supervised learnings. The most common DL-based algorithms are convolutional neural networks [70], deep Boltzmann machines [71], deep belief networks [72], autoencoders [73], recurrent neural networks [74] and long short-term memory networks [75]. In Figure 3.10 is presented a typical structure of a DNN model.

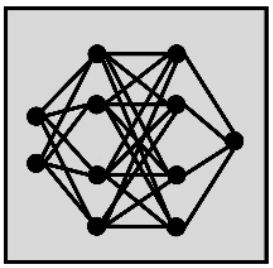


Figure 3.10 Deep neural networks model

3.6.7 Dimensionality Reduction Models

Dimensionality reduction (DR) based models aim of the models is to convert the original higher dimensional data set into lower dimensional representation to preserve as much information from the original data as feasible and to reduce the poor performance which is produced from the data sets with a large number of features. DR-based models can be used for supervised and unsupervised learning types and usually are applied to solve regression problems. The most common DR-based algorithms are principal components [76], partial least squares [77] and linear discriminants [78]. In Figure 3.11 is presented a typical structure of a DR model.

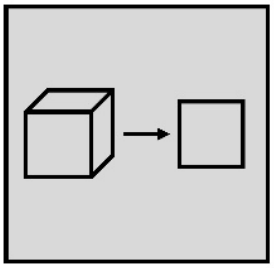


Figure 3.11 Dimensionality reduction model

3.6.8 Ensemble Learning Models

Ensemble learning (EL) models are designed to improve the prediction performance of a given statistical learning or model fitting technique by developing a linear combination of simpler base learners. So, each trained simpler base learner consists of a single hypothesis. EL-based models or multiple-classifier systems enable hybridization of hypotheses that were not produced by the same base learner, producing improved outcomes in the case of high variety among the single models. Typically, in EL-based models as the base learner is used the DT architecture. Common EL-based algorithms are AdaBoost [79], bootstrap aggregating [80], boosting technique [81], gradient boosting machines [82] and random forest [83]. In Figure 3.12 is presented a typical structure of an EL model.

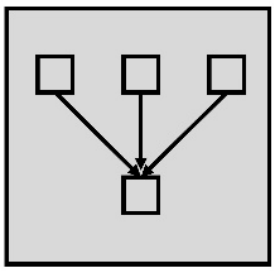


Figure 3.12 Ensemble learning model

3.6.9 Generative Learning Models

Generative learning (GL) models aim to generate new synthetic samples. A typical GL model consists of two neural networks, the generative network and the discriminative network. The generative network learns how to produce new synthetic samples according to the initial data set and the discriminative network distinguishes the generated from the initial original samples. GL-based models mostly are used to generate new samples in art, video games and advertising. Common GL-based algorithms are GANs [84], conditional generative adversarial networks (CGAN) [85], Wasserstein generative adversarial network WGAN [86], WCGAN [87], StyleGAN [88] and CycleGAN [89]. In Figure 3.13 is presented a typical structure of a GL model.

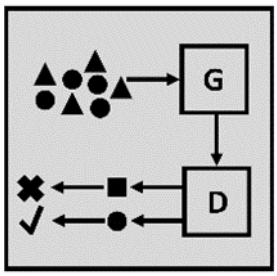


Figure 3.13 Generative learning model

3.6.10 Instance Based Models

Instance based (IB) models are memory-based models that learn from the comparison of new cases to instances in the training data set. These types of models construct hypotheses directly from the available data. Also, IB-based models generate regression or classification predictions only via specific instances while these models do not adhere to a set of abstractions. The main disadvantage of IB-based models is that their complexity increases with data. The most common IB-based algorithms are the k-nearest neighbor [90], vector quantization [91], locally weighted [92], support vector machines [93] and self – organizing map [94]. In Figure 3.14 is presented a typical structure of an IB model.

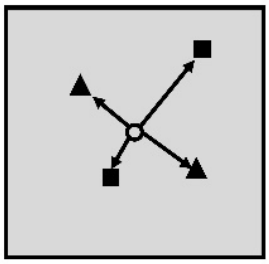


Figure 3.14 Instance based model

3.6.11 Natural Language Processing Models

Natural Language Processing (NLP) models are used to provide automatic summarization of the main points in a given text or document. NLP-based algorithms are also used to classify text according to predefined categories or classes and are used to organize information, and in email routing and spam filtering. The most common NLP-based algorithms are BERT [95] and XLNet [96]. In Figure 3.15 is presented a typical function of an NLP model.

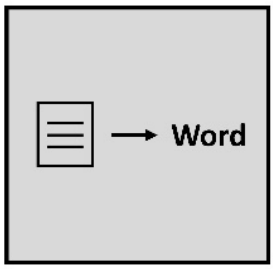


Figure 3.15 Natural language processing model

3.6.12 Regression Models

The goal of a regression learning model is to predict an output variable based on known input variables. The most common regression-based algorithms are linear regression [97], logistic regression [98], ordinary least squares regression [99], cubist [100] and locally estimated scatterplot smoothing [101]. In Figure 3.16 is presented a typical structure of a regression model.

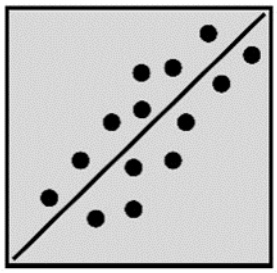


Figure 3.16 Regression model

3.6.13 Regularization Models

Regularization models consist of an extension of regression models. The aim of regularization-based models is through a penalize technique to simplify complex models to simpler performance models. Common regularization algorithms are ridge regression [102], least absolute shrinkage and selection operator [103] and least-angle regression [104]. In Figure 3.17 is presented a typical structure of a regularization model.

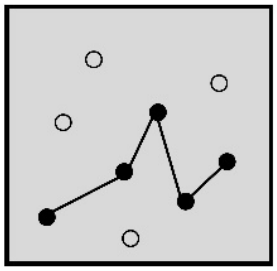


Figure 3.17 Regularization model

3.6.14 Speech Recognition Models

Speech recognition (SR) models or voice recognition models are used in speech recognition technology to convert voice to text. SR-based models work by breaking down the audio of a speech recording into individual sounds, analyzing each sound, using algorithms to find the most probable word fit in that language, and transcribing those sounds into text. Most common SR-based algorithms are ContextNet [105], LiGRU [106] and ResNet [107]. In Figure 3.18 is presented a typical function of an SR model.

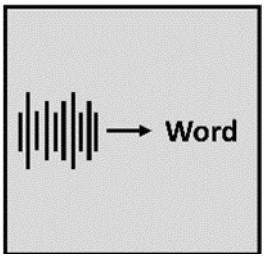


Figure 3.18 Speech recognition model

3.7 AI History Timeline

As can be observed from the Figure 3.19, the first algorithms were created in 1950 with the aim of developing simple AI models to solve basic mathematical problems. Moreover, from 1950-1970 an increase in the development of new algorithms can be observed. While from 1980-2000 there is a sharp decline. The main reason was the need to solve increasingly complex mathematical problems, combined with the lack of computational resources. This led to a lack of interest in this field of research. While, it is observed that since 2014, the period in which computing resources have increased, more sophisticated algorithms are being developed to solve more complex problems, such as computer vision, natural language processing and speech recognition problems.

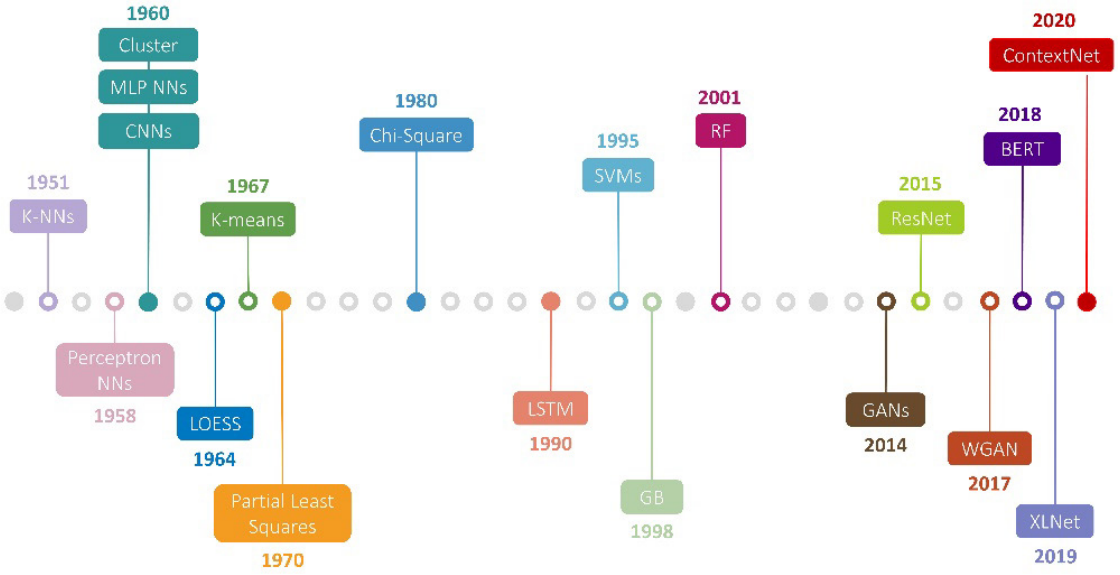


Figure 3.19 ML & DL algorithms history timeline

Chapter 4 Countermeasures Against Hardware Trojans

4.1 Introduction

As mentioned, HTs can be inserted at any stage and phase of ICs development, can attack at any unit of the ICs, can affect the ICs via a variety of attacks and can have different physical layouts. For these reasons in this thesis, we categorized the countermeasures approaches against HTs in three major categories, SCA-based approaches, ML-based & simulation approaches and auxiliary approaches (Figure 4.1). SCA-based approaches are categorized into two subcategories power and time analysis approaches. ML-based and simulation analysis approaches are also categorized into two subcategories Logic Testing (LT) and ML-based classification. And the auxiliary approaches are categorized in Runtime Monitoring (RM) and Prevention-Facilitation (PF) approaches.

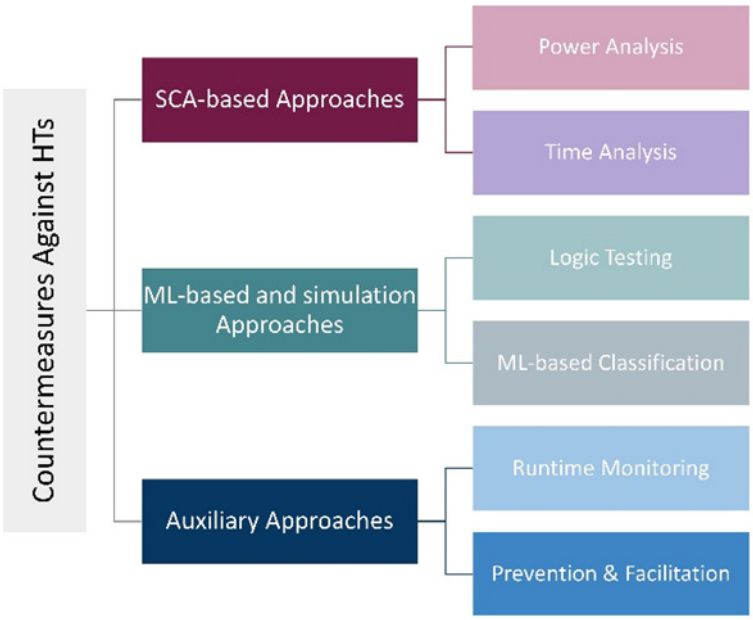


Figure 4.1 Categorization of countermeasures approaches against HTs

4.2 Historical Throwback

Historically, the first research attempt that mentioned and studied the existence of HTs in ICs was presented by Agrawal et al [108] in 2007. The authors have developed the first detection approach based on SCA-based power analysis. In 2009, Chakraborty et al [109] developed the first method for HT detection based on LT. In 2012, Salmani et [110] proposed

the first PF approach. In 2014 introduced by Bao et al [111] the first ML-based approach for the post-silicon stage. In 2015, Ngo et al [112] proposed an RM approach. Lastly, in 2016, the detection of HTs at GLN was proposed by Hasegawa et al [24], while in 2022 we proposed GAINESIS [113] the first GAN-based approach for the synthesis of new generated samples for GLN. In Figure 4.2 is presented a history timeline for countermeasures against HTs.

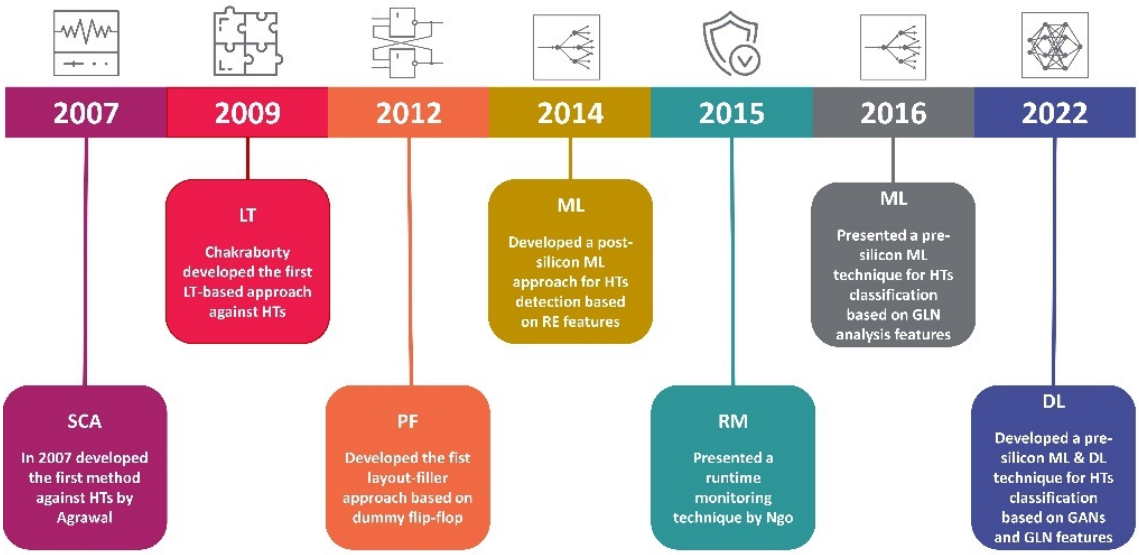


Figure 4.2 History timeline for countermeasures against HTs

4.3 Categorization of Studies

We present twenty-nine approaches in total. Twelve are conference and sixteen are journal articles referring to a period between 2007 and 2019 (Figure 4.3).

As mentioned, the category of SCA-based approaches consists of two subcategories SCA-based power analysis and SCA-based time analysis. Specifically, SCA-based power analysis consists of five approaches, four journals and one conference. While SCA-based time analysis has only two journal approaches. Next, the category ML-based and Simulation consists of thirteen approaches. LT simulation subcategory consists of three in total approaches, two journals and one conference. ML-based subcategory consists of ten in total approaches two journals and eight conference studies. While the last category consists of two subcategories RM and PF. RM-based subcategory consists of two journals and one

conference study. While the PF subcategory consists of four journals and two conference methods. (Figure 4.4).

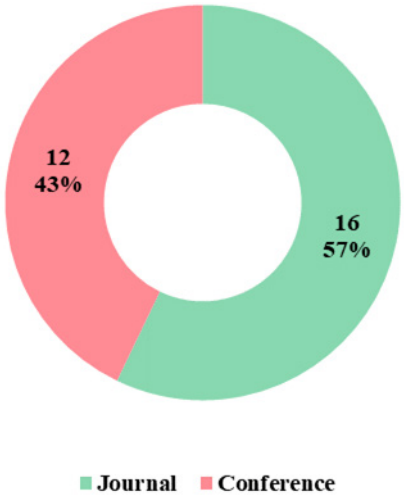


Figure 4.3 Categorization of studies

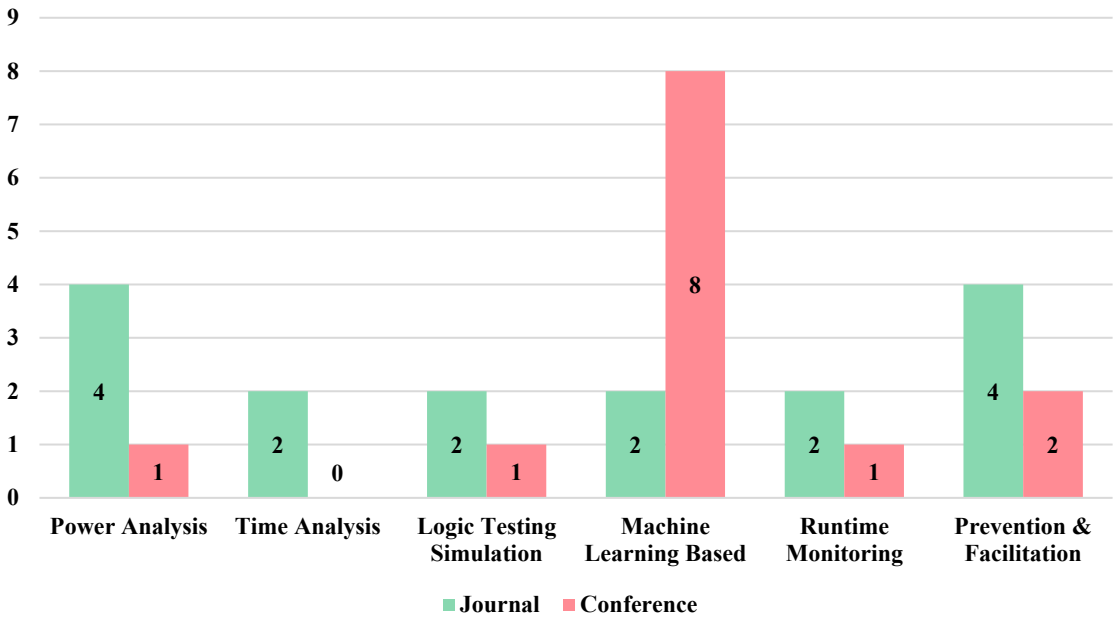


Figure 4.4 Categorization of studies per sub-categories

4.4 Distribution of the most Contributing Journal Studies

For these studies, it was noted that 50% were developed by academic institutions in the USA. Japan is in second place, with 13% of the total studies, which are based on ML at the GLN phase. China and Iran have 10% of the studies each. China is involved in the development of ML-based studies for the post-silicon stage of ICs. While Iran is dealing with the

development of studies for the PF subcategory. Furthermore, France, Austria, Malaysia and India have 3% each. (Figure 4.5).

Our next step was to present the contribution of the most important journals according to the examined studies. In total twenty-two journals and conferences were used for the publishment of the examined studies. 64% of the studies were published at conferences and 36% at journals. From the eight in total journals the most significant journals were the "Transactions on Information Forensics and Security", "Transactions on Very Large-Scale Integration Systems" and "Transactions on Computers" with three published studies each. Then follows the "Microprocessors and Microsystems" journal with two published studies. Specifically, "Transactions on Information Forensics and Security" journal published studies that mainly focused on SCA-based power analysis approaches. While the "Microprocessors and Microsystems" journal published studies that focused exclusively on PF approaches (Figure 4.6).

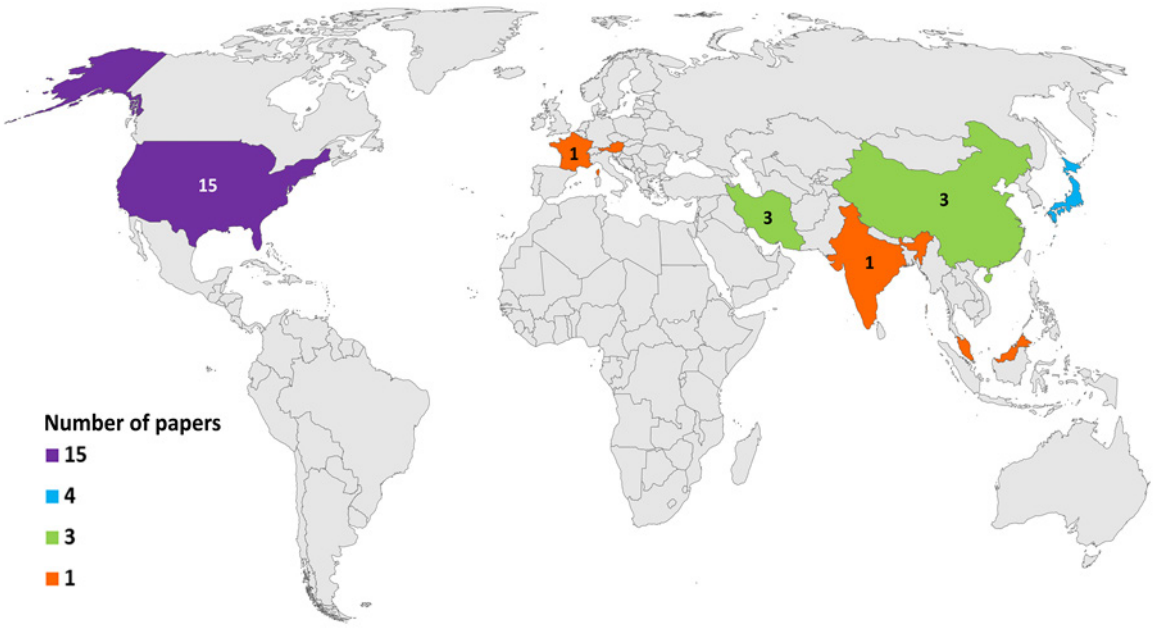


Figure 4.5 Geographical distribution of the contribution of each country to the research field focusing on countermeasures against HTs viruses.

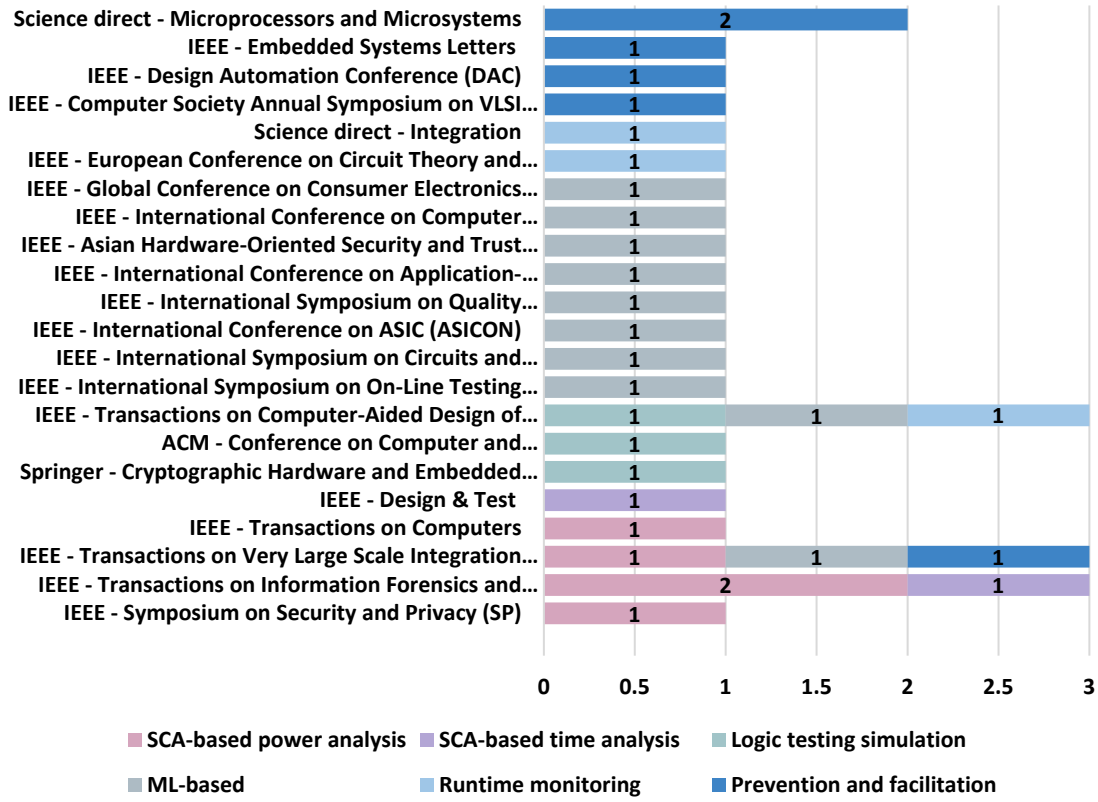


Figure 4.6 Distribution of the international journals and conferences and concerning applications of studies per sub-categories.

4.5 Studies Trend

In Figure 4.7 we can observe the popularity of each sub category over the years. Specifically, from 2007 to 2013 most of the studies focused on the development of methods for the detection of HTs based on SCA power and time analysis. In 2012 the first auxiliary-based study appears. And the golden era of auxiliary based approaches was 2015 when the majority of these studies are developed. The first ML-based approach was introduced in 2014. But in 2016 and 2017 there is a sharp increase in the development of such methods. As regards the LT simulation approaches the first study was presented in 2009 and other such approaches have been developed over time.

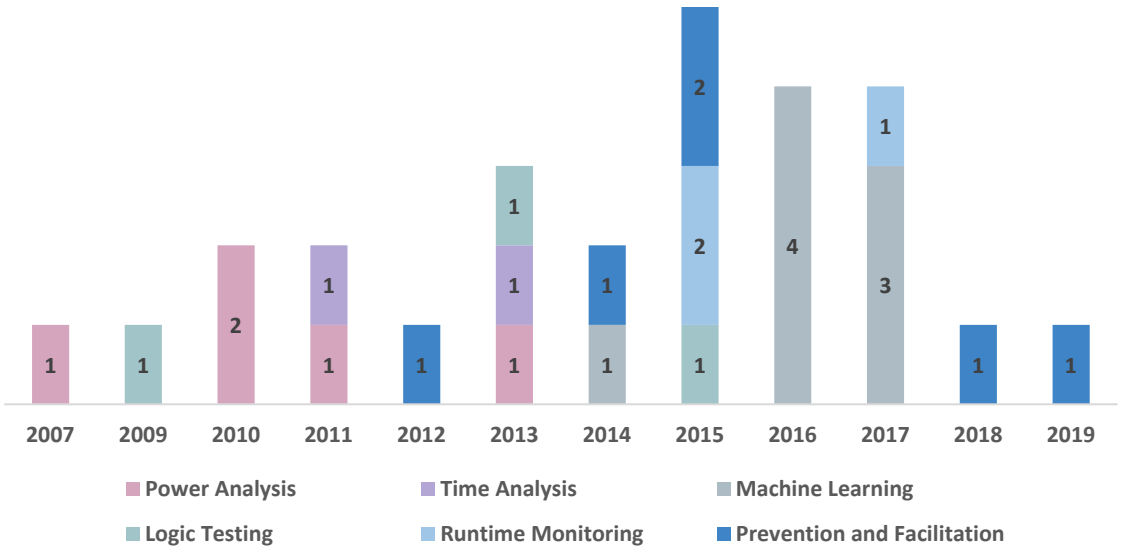


Figure 4.7 Countermeasures trend

4.6 SCA-based Approaches

SCA-based approaches aim to secure ICs for the SCA phase of the post-silicon stage of ICs. These approaches use techniques based on side-channel analysis features and detect changes of physical characteristics like power and time, caused by HTs. If the original SCA values of an IC differ, then the circuit is infected. That is caused because when an HTs is partially or fully activated the original infected circuit exhibit greater switching activity compared to the original normal circuit.

4.6.1 SCA-based Power Analysis Approaches

The first study which mentioned the existence of HTs was presented in 2007 by Agrawal et al. [108], and it was an SCA-based approach. Specifically, the authors developed a method for the detection of large or small in physical layout HTs based on SCA of transient current characteristics. In a study [114], the authors introduced a method for the detection of HTs based on SCA of static current characteristics. For multiple places across the 2-D surface of the chip, they took simultaneous measurements of static current features. The experimental results showed that this multiple measurement techniques in combination can effectively detect small HTs. Furthermore, authors in the study [115], proposed an SCA-based method via a power supply transient signals analysis. To evaluate local power supply transient signal measurements received from many individual power ports on the chip, a power supply

transient analysis technique was applied. The power supply transient signals for each power port were measured, and the power supply transient of each surrounding power port was compared. Following that, a signal calibration was used to reduce noise, and a scatter plot analysis was designed to detect an HT effectively. The final results showed that this technique was able to detect large physical layout HTs. In 2011 developed by Koushanfar et al. [116] a unified framework based on SCA leakage power. The authors also combined calibration and sensitivity analysis techniques for the detection of HTs. This approach was able to detect with low process overhead large in physical layout HTs. The last SCA-based on power features approach presented in this book is the study [117]. Specifically, the authors proposed a multiple-parameter SCA-based approach for the detection of HTs. They used and combined dynamic current and maximum frequency analysis features for HTs detection. The results showed that their approach was able to detect varying types and sizes of HTs. In Table 4.1 is presented a summary of SCA-based power analysis approaches.

Table 4.1 Summary of approaches in SCA-based power analysis

Authors	Observed Features	Feature Number	Functionality	Effectiveness	Benchmark	Type
[108]	Transient supply current (IDDT)	1	Detection of HTs in ICs, based on side-channel information analysis via transient current	Large and small HTs	RSA Circuit	Simulation
[100]	Quiescent supply current (IDDQ)	1	Detection of HTs based on the analysis of a chip's IDDQS	Small HTs	N/A	Experimental
[101]	Transient supply current (IDDT)	1	Detection of HTs via sensitivity analysis of power signal	Large HTs	ISCAS 85 Benchmark Circuit: C499	Simulation
[102]	Delay (T), Quiescent supply current (IDDQ), Transient supply current (IDDT)	3	Detection of HTs in ICs based on gate-level characterization and multi-parameter measurements	Large HTs	ISCAS 85 Benchmark Circuits: C8, C499, C432, C1355, C3450	Simulation
[103]	Transient supply current (IDDT), Maximum operating frequency (Fmax)	2	Detection of HTs, based on dynamic current and maximum operating frequency	Varying types and sizes of HTs	Xilinx FPGA: Virtex-II XC2V500	Simulation/ Experimental

4.6.2 SCA-based Time Analysis Approaches

In 2011 developed by Lamech et al [118] an SCA-based on time analysis features approach. Specifically, the authors combined SCA delay and power features for the detection of HTs. The experimental results showed that their method was able to detect large and small in size HTs. In 2013 Xiao et al. [119] developed an approach based on clock sweeping and SCA delay characteristics. They used a combination of path delay fault patterns with clock sweeping transition technique for the detection of HTs in a circuit. The results showed that their method could detect small in size HTs. In Table 4.2 is presented a summary of SCA-based time analysis approaches.

Table 4.2 Summary of approaches in SCA-based time analysis

Authors	Observed Features	Feature Number	Functionality	Effectiveness	Benchmark	Type
[103]	Power, Delay (T)	2	Detection of HTs, based on the analysis of power and delay	Large and small HTs	Xilinx FPGA Circuit: Virtex XUP-V2Pro	Experimental
[119]	Transition, Delay (T)	2	Detection of HTs based on clock sweeping and delay-based detection	Small HTs	ISCAS 89: S38417	Simulation/ Experimental

4.6.3 SCA-based Approaches Conclusions

Power analysis approaches constitute 71% of the total approaches in the SCA-based category. While the time analysis approaches 29% (Figure 4.8). As regards the benchmark, ISCAS 85 and custom circuits were the most used for power analysis approaches while ISCAS 89 and custom circuits for time analysis approaches (Figure 4.9). Finally, as far as features were concerned, the most used features for power analysis approaches were the quiescent and transient supply current and the delay for time analysis approaches (Figure 4.10).

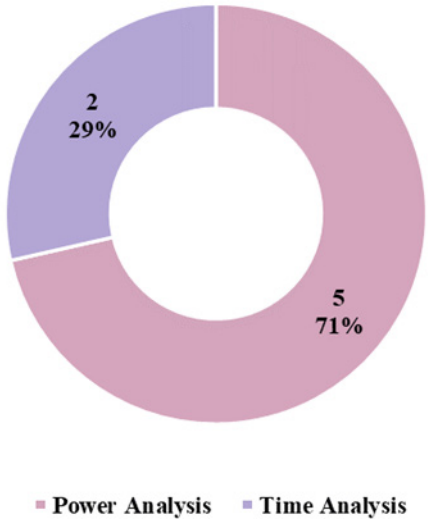


Figure 4.8 Number of studies in SCA-based approaches category

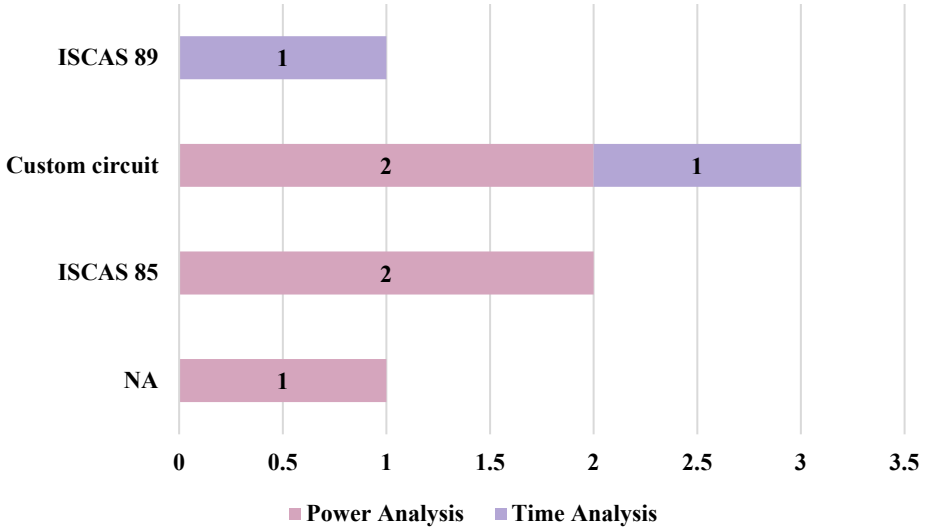


Figure 4.9 Benchmark in SCA-based approaches category

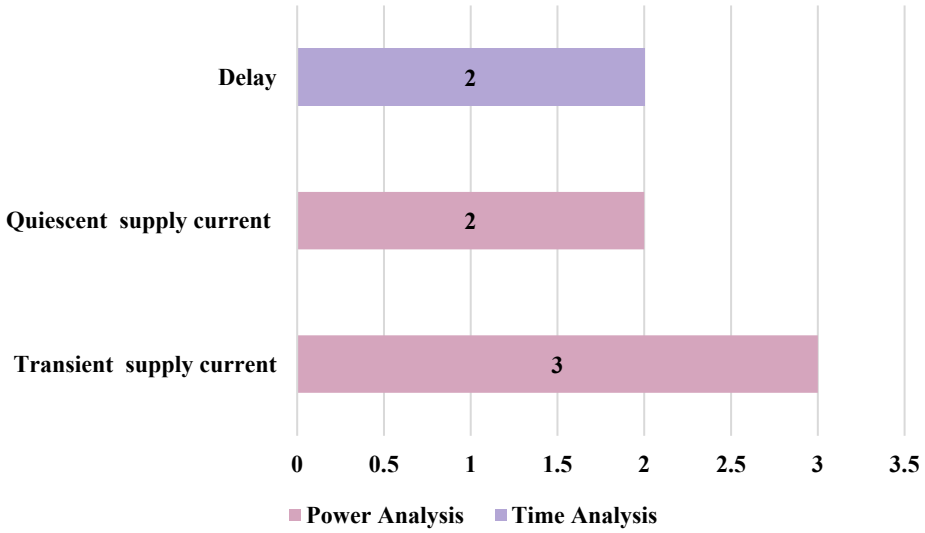


Figure 4.10 Features types in SCA-based approaches category

4.7 ML and Simulation based Approaches

ML approaches aim to handle HTs based on classification. These types of approaches developed ML-based classifiers for the classification of HTs in different phases of ICs development. On the other simulation-based approaches like logic testing techniques aim to generate tests that activate HTs and propagate the HTs payload to primary outputs for comparison with the golden circuit. The challenge with these techniques is to generate efficient tests to activate HTs. In this section, are presented ML and simulation-based approaches as countermeasures against HTs.

4.7.1 Logic Testing Simulation Approaches

As mentioned, LT simulation approaches aim to generate effective tests in order to be able to activate and discover the stealthy nature of HTs. Due to the stealthy nature of HTs it is difficult to distinguished an infected circuit. Random generated tests are not efficient for this reason the LT-based simulation approaches aim to generate guided tests for the activation and detection of HTs. In 2009 Chakraborty et al. [109] proposed an approach based on LT simulation as a countermeasure against HTs. Specifically, they developed an LT approach named MERO. This approach generated test patterns based on multiple excitations of rare logic conditions at internal nodes. The simulation results showed that this approach was able to detect small in size HTs. In 2011 Waksman et al. [120] developed an LT-based framework named FANCI. They used Boolean functional analysis features to generate test patterns for HTs activation. The results showed that this approach was able to detect infected circuits with a low false positive rate. In the last study [121], the authors developed an LT-based

simulation technique named VeriTrust for the detection of HTs at the design phase based on HTs trigger inputs. VeriTrust technique consisted of a traced and a checker. The tracer parsed verification tests to identify trigger signals containing inactive entries while the checker examined these signals to determine which are associated with HTs. The results showed that this approach was able to detect different types and sizes of HTs. In Table 4.3 is presented a summary of LT simulation approaches.

Table 4.3 Summary of LT simulation approaches.

Authors	Observed Features	Feature Number	Functionality	Effectiveness	Benchmark	Type
[109]	Nodes	1	Detection of HTs based on test pattern generation and multiple excitations of rare logic conditions at internal nodes	Small HTs	ISCAS 85: C2670, C3540, C5315, C6288, C7552 ISCAS 89: S13207, S15850, S35932	Simulation
[120]	Wires	1	Detection of HTs based on Boolean functional analysis	HTs and IPs	ISCAS 89: S15850, S35932, S38417	Simulation
[121]	Netlists	1	Identification of HTs at the design stage, based on the detection of trigger inputs	Different types and sizes of HTs	ISCAS 89: S15850, S35932, S38417, S38584 Microcontrollers: MC8051, LEON3	Simulation

4.7.2 ML-based Approaches

ML-based approaches aim to detect the existence of HTs in a circuit. In these approaches are developed models which can classify infected from normal circuits or to use as reverse engineering or side-channel analysis methods for the detection of HTs in a circuit. Specifically, for the pre-silicon stage proposed ML-based classifiers for the classification of infected and normal circuits at different pre-silicon phases. While ML-based methods that work as reverse engineering techniques and ML-based methods trained via side channel analysis features were developed for the detection of HTs at the post-silicon stage.

For the pre-silicon phase in 2016, Hasewaga et al. [24] proposed an SVM-based model for the classification of infected from normal circuits. Specifically, the authors developed an SVM-based model for the classification of HTs at the GLN phase of the pre-silicon stage. For the training of the model was used a data set consisting of GLN-based features like nets

and gates of the circuits. The results showed that this approach was able to classify effectively the infected with HTs from normal nets. The same group [25] 2017 proposed another ML-based model. They developed an RF-based model which was trained via GLN-based area features, like number of flip-flops and multiplexors before and after for each net. The results showed that the RF-based model was effective for the classification of the two classes. In 2018 Inoue et al. [26] proposed an SVM-based model in a combination with GLN-based area features for the classification of HTs at the GLN phase of the pre-silicon stage of ICs development. The SVM-based model was trained via area features like the number of logic gates and flip-flops for each net of the infected and normal circuits. The final results proved the validity of the method. In the study [27], the authors developed six ML-based models for the classification of HTs at the GLN phase. Specifically, they developed and compared six ML-based models which were trained via a dataset consisting of GLN-based area, power, and time analysis features from infected and normal circuits. The features consisted of area features like the number of cells, nets, ports, and power features like the number of total switching and combinational power of each normal and infected circuits. The experimental results showed that their GB-based model was able to classify effectively the normal from HTs circuits.

As mentioned, also ML-based approaches were developed for the detection of HTs at the post-silicon stage. So, for the post-silicon phase in 2014 Bao et al. [111] developed an ML-based model as a reverse engineering approach for the detection of HTs. Specifically, they trained an SVM classifier based on high resolution images features from golden and infected with HTs circuits layouts. The simulation results showed that the SVM-based classifier was able to classify the two classes efficiently. The same group in the study [122] proposed a KMeans-based clustering model. The KMeans-based model has developed again via high resolution image features from golden circuits and of three types of modifications based on the golden circuits which consisted of the infected circuits. Another post-silicon detection approach was developed in 2016 by Jap et al. [123]. Specifically, the authors developed an SVM-based model for the detection of HTs. The model was trained from a data set consisting of SCA-based time features like leakage from normal and infected circuits. Another study with ML and SCA techniques was proposed by Xue et al. [124]. In this study, the authors developed an SVM-based model for the detection of HTs at the post-silicon stage. The model was trained via a data set that consisted of SCA-based power features and specifically transient power supply features of normal and infected circuits. The experimental results

showed that this method was able to detect with effectiveness the infective from normal circuits. Wang et al. [125] proposed another SCA-based method in combination with ML techniques for the detection of HTs at the post-silicon phase. Specifically, they developed an ELM-based model which was trained from a data set consisting of dynamic power features from infected and normal circuits. In the study [126], the authors developed an SVM-based model for the detection of HTs via SCA power features. Specifically, they developed an SVM-based model which was trained via a data set consisting of SCA-based power consumption waveforms features from infected and normal circuits and given. The experimental results proved the validity of the method. Liu et al. [127] proposed another SCA-based in combination with an ML-based model approach for the detection of HTs at the post-silicon phase. They developed an SVM-based model which was trained via SCA wireless transmission power waveform features from HTs free and infected circuits. The results showed that their method was able to detect effectively wireless transmissions power signals produced from HTs. In Table 4.4 is presented a summary of ML-based approaches.

Table 4.4 Summary of ML-based approaches

Authors	Observed feature	Feature number	Positive data	Negative data	Benchmark	Models/ Algorithms	Results
[24]	Features extracted from known gate-level netlists, like LGFi, FFi, FFo, PI and PO	5	118.969 Trojan Nets	121.452 Normal Nets	Trust-HUB: RS232-T1000, RS232-T1600, S15850-T100, S35932-T100, S35932-T300, S38417-T100, S38417-T300, S38584-T100, S38584-T300	SVM	80% - 100% TPR
[25]	Features extracted from gate-level netlists,	11	429 Normal Nets	54.782 Normal Nets	Trust-HUB: RS232-T1000, RS232-T1200, RS232-T1300, RS232-T1400, RS232-T1500, S15850-T100, S35932-T100, S35932-T300, S38417-T100, S38417-T200, S38417-T300, S38584-T100	EL/RF	74.6% F-measure
[26]	Features extracted from netlists, like LGFi, FFi, FFo, PI and PO	5	248 Trojan nets	1.991 Normal nets	Trust-HUB: RS232-T1000, RS232-T1100, RS232-T1200, RS232-T1300, RS232-T1400, RS232-T1500, RS232-T1600	SVM	Type A: 58.9% accuracy Type B: 69.5% accuracy Type C: 65.1% accuracy

[27]	Features from area, power and time analysis through DC compiler tool	11	892	18	Trust-HUB: All- Benchmarks	GB	100% F1-score
[111]	High resolution images from ICs golden layouts	160x160 pixels	500 Trojan Addition 500 Trojan Deletion 500 Trojan Parametric	500 Trojan Free	Custom ISCAS 89: S27, S298, S280, S15850, S38417 ITC 99: B18	SVM	90% accuracy
[122]	Trojan Free ICs golden layout images and 3 types of modifications produced based on these images, Trojan Addition, deletion and parametric	160x160 pixels	500 Trojan Addition 500 Trojan Deletion 500 Trojan Parametric	500 Trojan Free	Custom ISCAS 89: S27, S298, S280, S15850, S38417 ITC 99: B18	Clustering/ K-Means	Trojan-Free: 99.23% accuracy Trojan-Addition: 100% accuracy Trojan-Deletion: 100% accuracy Trojan-Parametric: 98.86% accuracy
[123]	Features extracted from side-channel analysis to leakage of the chip based on time samples	4	N/A	75.000 Time samples	Xilinx FPGA Circuit: Spartan-6	SVM	N/A
[124]	Features extracted from the transient power supply currents (IDDT) of each simulated IC and a Trojan-free or Trojan-inserted indicator	501	50 Trojan Infected	50 Trojan Free	ISCAS 89: S38417, S35932	SVM	Trojan-inserted ICs known: 100% accuracy Trojan-inserted ICs unknown: 98% accuracy

[125]	Features from converted power consumption waveform into the frequency domain	N/A	N/A	N/A	N/A	SVM	72.72% accuracy
[126]	Features from side-channel analysis, dynamic power consumption	N/A	N/A	N/A	N/A	ANN/ELM	90% success rate
[127]	Features consist of transmission power measurements for six ciphertext blocks transmitted by each of 40 Trojan-free circuits	6	40 Trojan-I infected 40 Trojan-II infected	30 Trojan Free	Trojan-Free: TSMC Microcontroller: 0.35-µm technology Trojan-I and Trojan-II: Created two HTs, which leak the secret key of a wireless cryptographic IC consisting of an Advanced Encryption Standard (AES) core and an ultra-wideband (UWB) transmitter (TX).	SVM	0/10 FP and 0/80 FN

4.7.3 ML and Simulation based Approaches Conclusions

ML-based approaches constitute 77% of the total approaches in the ML and Simulation category. While the LT simulation approaches 23% (Figure 4.11). As regards the benchmark, Trust-HUB and ISCAS 89 were the most used for ML-based approaches while ISCAS 89 and ISCAS 85 for LT simulation approaches (Figure 4.12). Finally, as far as features were concerned, the most used features for ML-based approaches were netlists as well as high resolution images and dynamic power. While for LT simulation approaches were netlists, wire and nodes features (Figure 4.13).

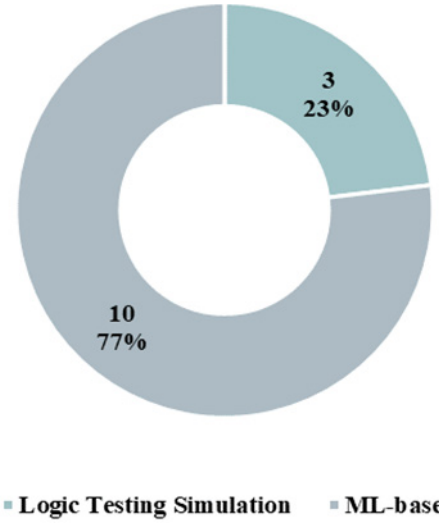


Figure 4.11 Number of studies in ML and Simulation based approaches category

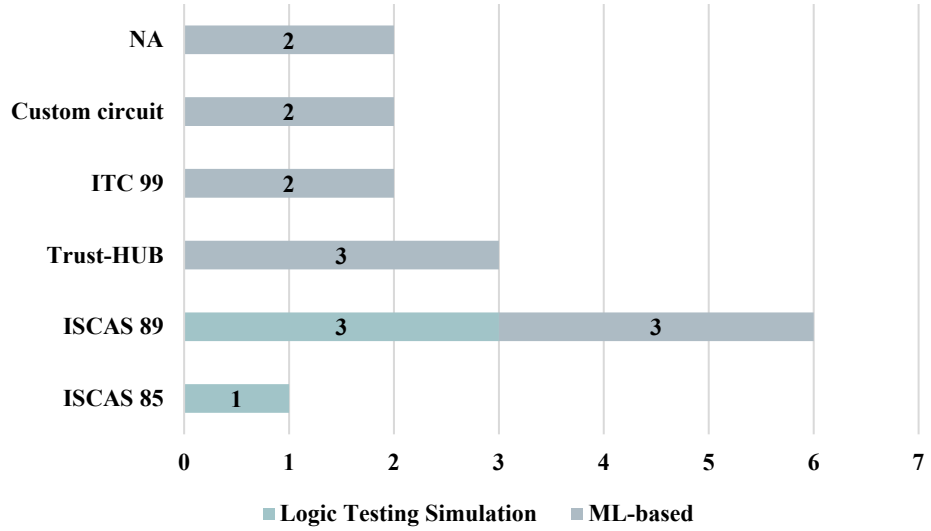


Figure 4.12 Benchmark in ML and Simulation based approaches category

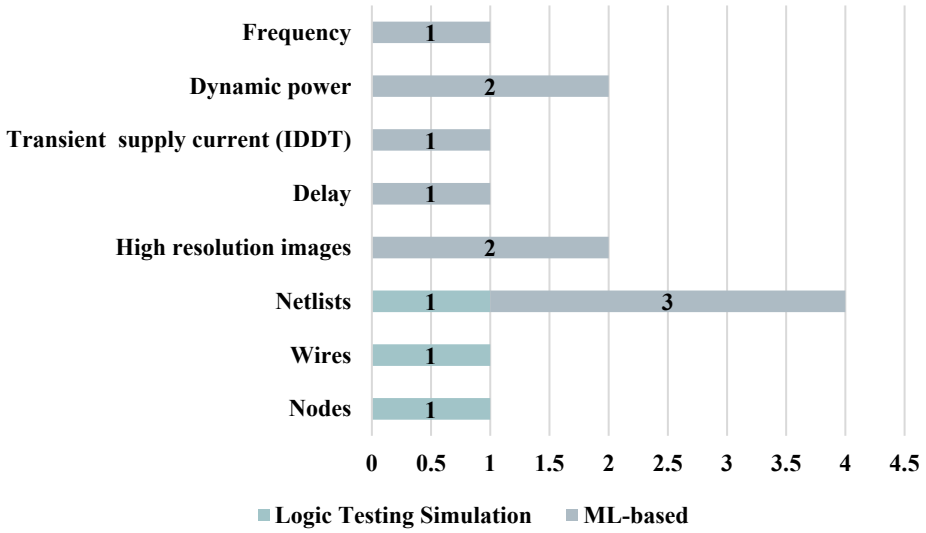


Figure 4.13 Features types in ML and simulation-based approaches category.

4.8 Auxiliary Approaches

The auxiliary approaches aim to enhance the effectiveness of the detection techniques against HTs for the pre-silicon or post-silicon stage. Auxiliary approaches can be categorized into two categories, the runtime monitoring approaches and the prevention-facilitation approaches.

Runtime monitoring approaches aim to reduce the catastrophic effects of HTs when these viruses are activated. Specifically, these approaches focus on identifying putatively undetectable attacks and their effects from time-delayed HT activation. These studies can develop techniques that can probe the behavior of signals of interest using finite state machines or can generate and run multiple functionally equivalent tests to detect HT attacks. Furthermore, these studies can find similar HTs, due to their parallel execution on the circuit or to bypass HTs, imitating software HTs. Also, runtime monitoring approaches can detect unused circuitry and label it as suspicious using verification tests. Subsequently, suspicious circuitry is replaced with a software logic exception which allows the normal performance of the system to bypass the HTs.

On the other, prevention-facilitation approaches aim to increase the difficulty for HT insertion into ICs, mainly during the design phase, or facilitate the detection approaches. Prevention-facilitation approaches use hardware security techniques like obfuscation, layout-filler, path-delay fingerprinting to enhance the detection of HTs. The obfuscation technique changes the transition mode of the circuit providing the ability to operate in two

different modes; normal and obfuscated. The normal mode produces the desired output for the circuit, while the obfuscated allows the circuit to malfunction in some of the input patterns. The use of this technique makes the insertion of a malicious circuit into a system more difficult. Layout-filler techniques are used to fill the empty spaces of a circuit with filler cells to prevent the insertion of additional components. However, these techniques cannot prevent the malicious conversion of a transistor set or the addition of a circuit that does not require additional layout space. Another way to detect HTs is based on synthesis algorithms based on path-delay fingerprints. These techniques improve the HTs detection probability by minimizing the maximum delay shortest path of the circuits.

4.8.1 Runtime Monitoring Approaches

In 2015 Ngo et al. [112] developed a runtime monitoring approach for the detection of HTs. Specifically, they developed an assertion approach for identifying and validating high-level important behavioral invariants through an integrated on the circuit, hardware property checker. The results showed that this approach could detect HTs in circuits with varying system overhead and modify the protection levels correspondingly. In the study [128], the authors developed a general methodology based on runtime monitors for the identification and detection of HTs attacks through burst mode communication. Specifically, they designed a runtime monitor approach based on the analysis of vulnerable paths. The statistical and experimental analysis showed that this technique had low area and power overhead compared to other monitor approaches and could easily be used without requiring extra information of IP modules. Furthermore, authors in the study [129], developed three low-overhead runtime approaches based on power/thermal features of infected and normal circuits for the detection of HTs. The first approach was a sensor-based approach based on thermal features extracted from the thermal sensors. In the second approach was used a filter known as the Kalman filter for the tracking of circuits thermal profiles. The third approach combined the Kalman filter with leakage power features of the circuits to track the thermal profiles. The simulation results verified that all the approaches were able to detect HTs effectively. In Table 4.5 is presented the summary of RM approaches.

Table 4.5 Summary of RM approaches

Authors	Observed Features	Feature Number	Functionality	Benchmark	Type

[112]	Critical behavioral invariants	Features number adapt according to the circuit	Configurable Security Monitor	Microcontroller Circuit: LEON3	Simulation
[128]	Handshaking protocol features	Features number adapt according to the circuit	Configurable Security Monitor	Trust-HUB: AES-T100, AES-T1000, AES-T1100, AES-T1200, AES-T1300, AES-T1400, AES-T1500, AES-T200, AES-T2000, AES-T2100, AES-T300, AES-T400, AES-T500, AES-T600, AES-T700, AES-T800, AES-T900	Experimental
[129]	Thermal and power profiles	2	Variant-Based Parallel Execution	Trust-HUB: AES-T1700, BasicRSA-T200, MC8051-T300, MC8051-T400, MC8051-T600, RS232-T400, RS232-T900, S38417-T300, PIC16F84-T100, PIC16F84-T200	Simulation

4.8.2 Prevention & Facilitation Approaches

An obfuscation-based technique was developed by Kamali et al. [130]. The authors developed an obfuscation-based method via embedded key features for the protection of ICs against HTs attacks. The simulation results showed that their method could defend ICs effectively. The same group in the study [131] proposed again an obfuscation-based method for the defense of IP-piracy and reverse engineering approaches via the replacement of parts of logic design with programmable logic routing blocks. In 2012 Salmani et al. [110] developed an improving HTs detection technique based on analysis of the transition generation time and dummy flip-flop insertion. Specifically, the authors developed a method based on dummy multiplexors to be able to remove rare trigger conditions, reduce the transition generation time, and increase the activity of HTs for the detection of HTs. In the study [132], the authors proposed a layout-filler based on a dummy circuit insertion technique against HTs attacks. This technique is identified and replaced the unused resources of a circuit with dummy logic cells. Experimental results showed that the proposed study was effective for Field Programmable Gate Arrays (FPGAs) with no cost on power or performance. In 2014 Nejat et al. [133] developed an approach for improving HT detection based on path-delay fingerprinting and an effective test-vector selection scheme. The fundamental idea behind this method was to test the circuit at the appropriate frequencies. Each path was examined at a clock cycle with a period equal to the path's delay. The results showed that this method improves the detection of HTs with low area overhead. The same group in the study [134], developed a path-delay fingerprinting-based method for the

detection of HTs. Specifically, they developed a logic-level synthesis retiming algorithm that shortened for each node of a circuit the connection paths to minimize the communication delay. The results showed that the shorted paths improve the detection of HTs. In Table 4.6 is presented the summary of PF approaches.

Table 4.6 Summary of PF approaches

Authors	Observed Features	Feature Number	Functionality	Benchmark	Type
[130]	Several embedded key numbers	1	Obfuscation	ISCAS 85: C2670, C3540, C5315, C6288, C7552	Simulation
[131]	Fully programmable logic and routing blocks	2	Obfuscation	ISCAS 85: C432, C499, C880, C1355, C1908, C2670, C3540, C5315, C7552	Simulation
[110]	Features based on average clock cycles per transition	1	Dummy Circuit Insertion	ISCAS 89: S38417	Simulation
[132]	Low-level dummy logics (LLDLs)	N/A	Layout Filler	Xilinx FPGA Circuit: Virtex-II	Experimental
[133]	Features based on path-delay fingerprinting	1	Improvement of HT detection based on path-delay fingerprinting and an effective test-vector selection scheme	ISCAS 89: S713, S1423, S5378, S13207, S35932	Experimental
[134]	Features based on path-delay fingerprinting	1	Enhance HTs detection based on the improvement of the path-delay fingerprinting technique via a logic-level synthesis retiming algorithm	ISCAS 89: S208, S344, S1196, S1238, S1494, S9234, S13207, S38417	Simulation/ Experimental

4.8.3 Auxiliary Approaches Conclusions

PF approaches constitute 67% of the total approaches in the Auxiliary category. While the RM approaches 33% (Figure 4.14). As regards the benchmark, Trust-HUB and custom circuits were the most used for PF approaches, while ISCAS 89 and ISCAS 85 for RM approaches (Figure 4.15). Finally, the most used feature for PF approaches was the delay. While for RM approaches the most used features were thermal power, handshaking protocol and behavioral invariants features (Figure 4.16).

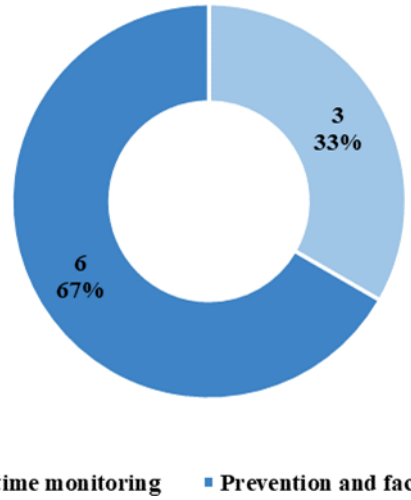


Figure 4.14 Number of studies in Auxiliary based approaches category

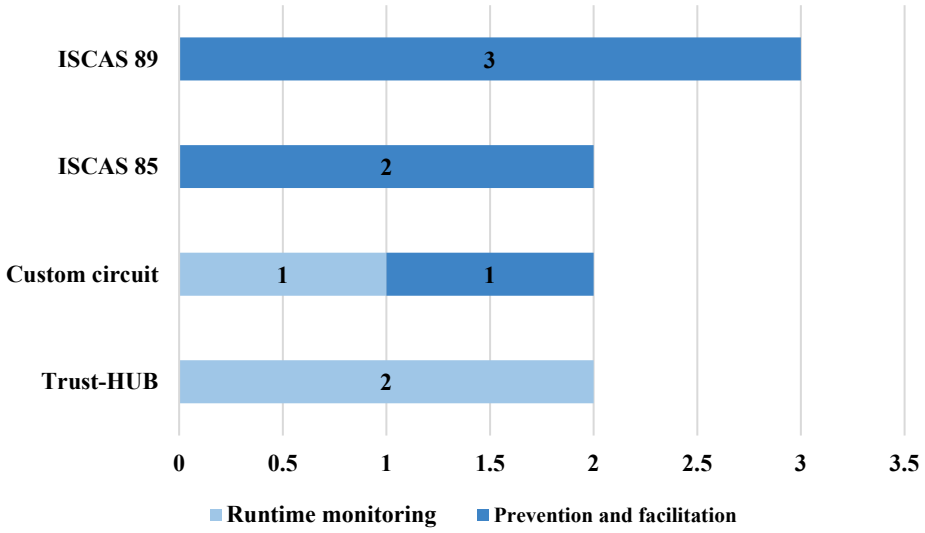


Figure 4.15 Benchmark in Auxiliary based approaches category

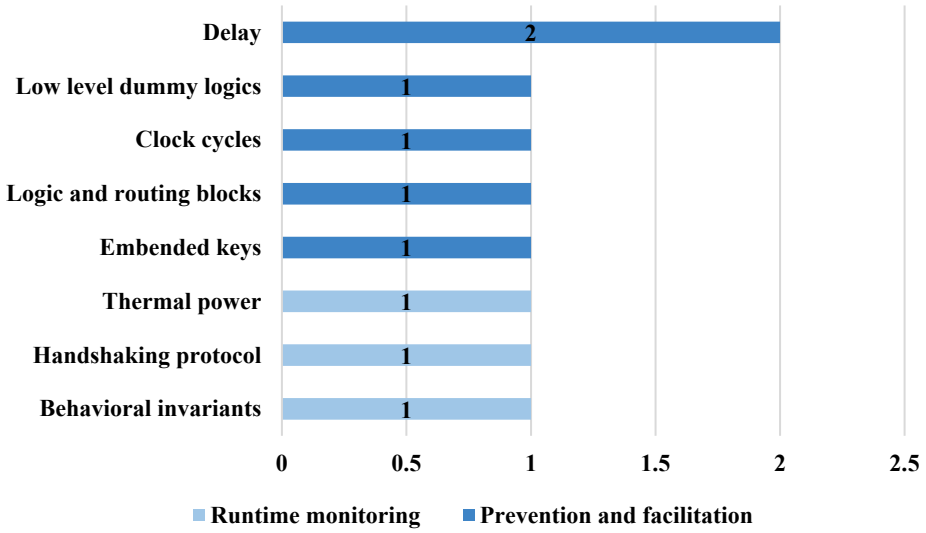


Figure 4.16 Features types in Auxiliary based approaches category

4.9 Countermeasures Against Hardware Trojans Conclusions

Since 2007, HT detection techniques have emerged as necessary tools for maintaining the reliable, secure and highly stable operation of virtually every available IC type. Depending on the underlying mechanism, the functionality and the manufacturing phase at which an HT detection method operates, we have grouped available techniques in three categories. Each category was further subdivided depending on specific functionalities related to the detection process. SCA-based approaches consist 24% (7 out of 29) of the total examined approaches. On the other, ML-based and simulation approaches consist 45% (13 out of 29) of the total approaches while the Auxiliary approaches consist of the remaining 31% (9 out of 29) (Figure 4.17).

As regards the benchmark, ISCAS 89 was the most frequently utilized benchmark with 34% (10 out of 29) of the total amount of studies. ISCAS 89 is especially used from ML-based, LT simulation and PF approaches. The next most used benchmark was the custom circuits with 24% (7 out of 29) and used especially for SCA-based power analysis and ML-based approaches. While the next most used benchmark was ISCAS 85 and Trust-HUB with 17% (5 out of 29) respectively. ISCAS 85 is mostly used for SCA-based power analysis and auxiliary PF approaches and Trust-HUB from RM and ML-based approaches (Figure 4.18).

Depending on the mode of operation and functionality, HT detection studies relied on a wide spectrum of features for training their models or extracting decision making rules. Delay, netlist and transient supply current were the most frequently used features. Specifically, the

delay was the most frequently used feature with 17% (5 out of 29) and was used mostly for SCA-based time analysis and PF approaches. While netlist and transient supply current were the next most used features with 14% (4 out of 29) respectively. Netlists are used exclusively for ML-based approaches and transient supply current for SCA-based power analysis approaches. The remaining studies depended on high-resolution imaging, dynamic and quiescent supply current power (Figure 4.19).

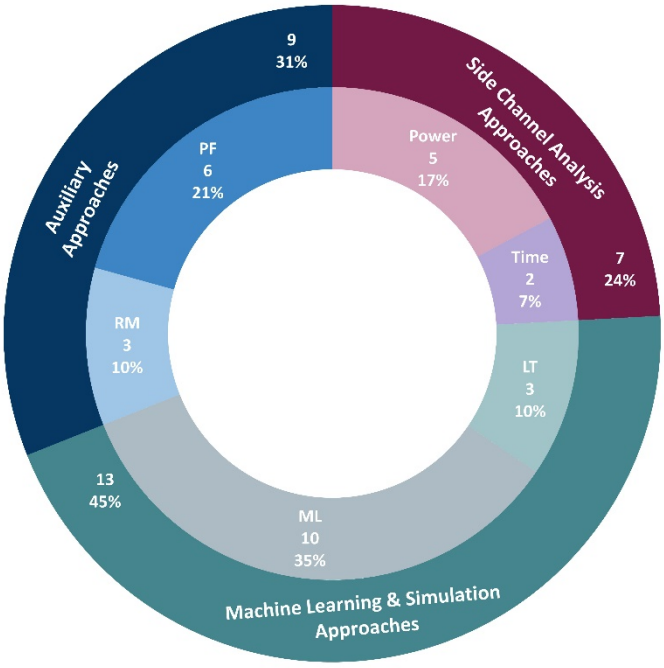


Figure 4.17 Number of studies for all the categories

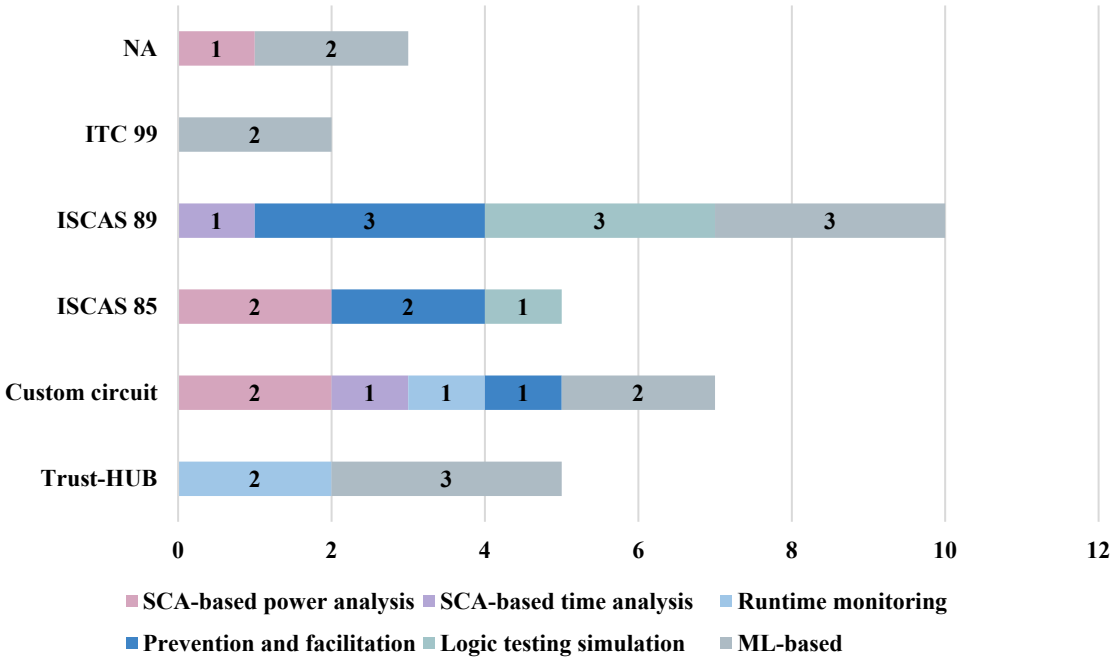


Figure 4.18 Benchmark for each countermeasure category

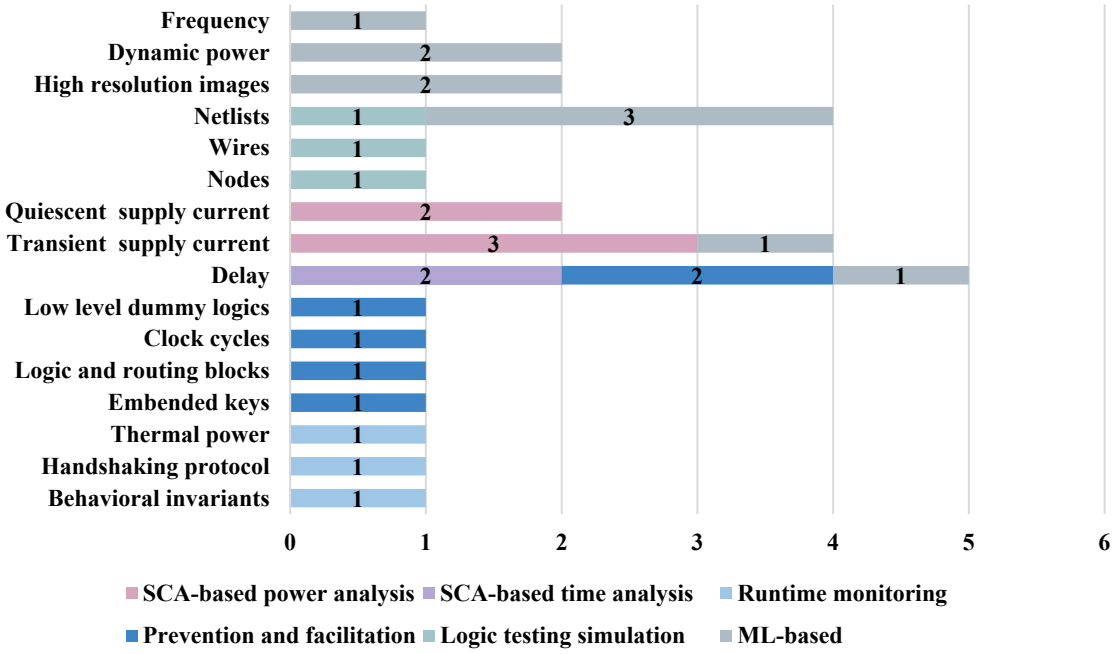


Figure 4.19 Features types for each countermeasure category

Chapter 5 GAINESIS: Generative Artificial Intelligence NEtlists SynthesIS

5.1 Introduction

In this chapter, we present our methodology. We list the steps needed to create an ML-based model as well as we mention the importance of the data set and the features. We present with details our scheme and finally, we analyse each step of our scheme in detail.

Must be mentioned that developing a model based on the principles of ML or DL is a costly process in both time and computing power. Depending on the problem, the size of the data set, the size, type and quantity of features contained in the data set as well as the algorithms, and the set of parameters that will be used and combined for the development of the new ML or DL-based model, time and computing power can vary significantly from model to model. Until today, ML-based models need significantly more time for training and testing than DL-based models. The reason is that ML-based algorithms for the development of a model are built to use the Computer Process Unit (CPU) and not the Graphic Process Unit (GPU). On the other, DL-based algorithms can use either CPU or GPU for the training and evaluation of the development model. In Figure 5.1 we present the steps for the development of an ML or DL-based model.

The development of GAINESIS was based on Python v3.6 [135] 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 [136], Keras v2.0 [137], Scikit-learn [138], the XGBoost library [139] and Jupyter Notebook [140] environment was used to develop all the tested GAN and ML models.

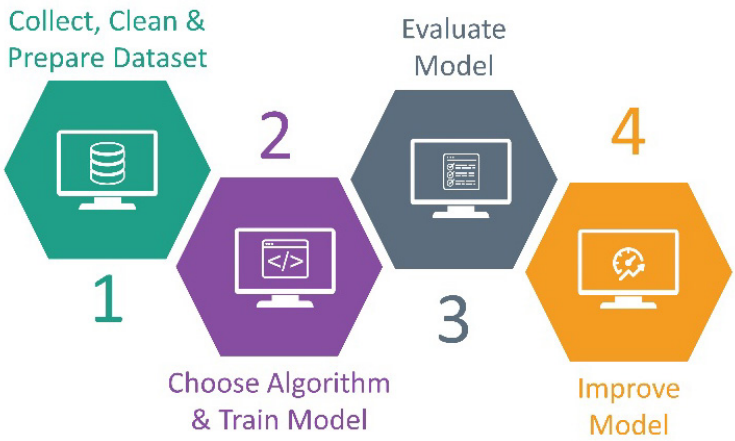


Figure 5.1 Steps for the development of an ML or DL-based model

5.2 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 [82], k-nearest neighbors (KNN) [90], logistic regression (LR) [98], multilayer perceptron (MLP) [41], RF [83], SVM [93] and XGB [139]. 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 [84], CGAN [85], WGAN [86], and WCGAN [87] 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 was 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 next 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). Finally, our last step was to compare our best GB-WCGAN-Mixed-600 classifier with existing methods (25). Our scheme is illustrated in Figure 5.2. 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.

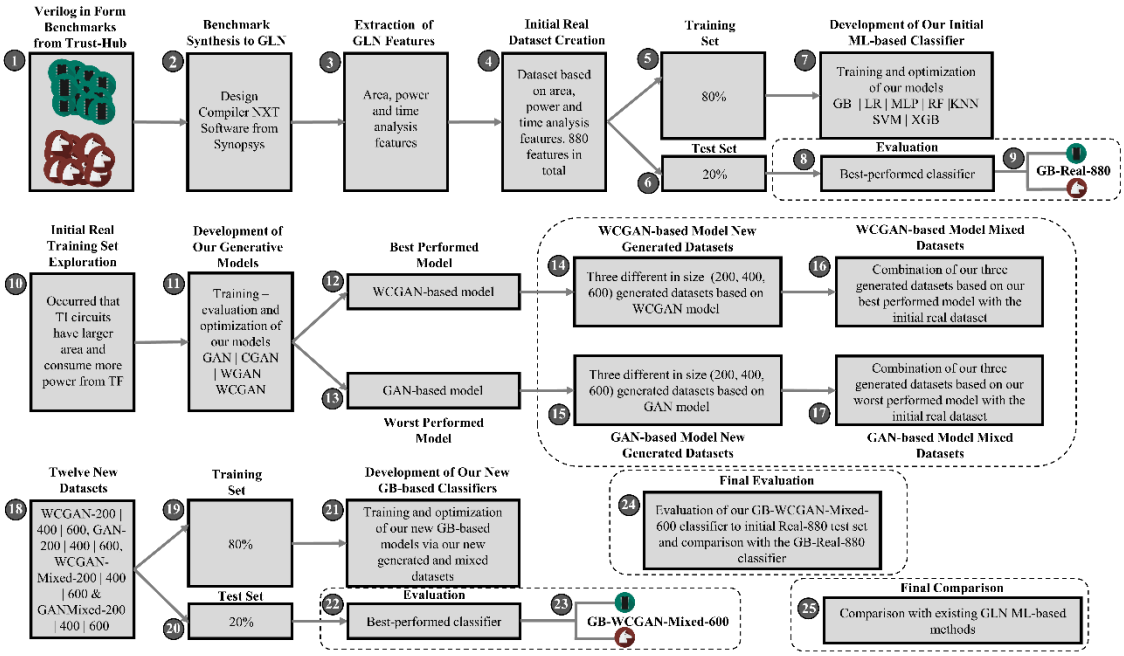


Figure 5.2 Scheme of our Artificial Intelligence-based approach for safeguarding integrated circuits at gate-level netlist phase against hardware Trojans, GAINESIS.

5.3 Data set

Every year more and more ML/DL-based approaches are developed as countermeasures against HTs. These approaches are aimed at classifying or detecting infected with HTs circuits from normal uninfected circuits. Also, some approaches are used to enhance the classification or detection methods. So, the development of these types of approaches needs a quality data set that will contain a sufficient number of quality samples and features to be able to train the ML/DL-based model more efficiently.

The data set can be divided into three categories, structured, unstructured and semi-structured. Structured data is data that follows a pre-defined data model and is thus easy to analyze. Structured data is presented in a tabular format, including relationships between rows and columns. Excel files and SQL databases are common examples of structured data. Each of them has sortable organized rows and columns. Unstructured data is information that lacks a predefined data model or is not organized in a specific way. Common examples of unstructured data include text, image, video or audio files. Semi-structured data is a type of structured data that does not follow the rules of structured data. However, tags or other markers are used to distinguish semantic pieces and enforce hierarchies of records and fields inside the data. Examples of semi-structured data include JSON and XML files.

As mentioned, the data set plays a significant role in the development of a robust ML- or DL-based model. Specifically, the data set before being used for the development of a model, must be cleared from unnecessary values and organised. For example, the data set must be checked for consistency, cleared of zeros and/or unspecified values, and labeled where needed. An unreliable data set like a data set with imbalanced samples per class leads to the development of unreliable models. A type of unreliable model is a model that was learned to over-classify a class compared with another class. Due to the lack of samples for a class, the model has learned to under-classify this class compared with the other one.

Must be mentioned that each sample or feature represents a measurable piece of data that can be used for analysis. The features which are included in a data set can vary widely depending on the problem which is analyzed. Features are the basic building blocks of the data set. The quality of the features in a data set has a major impact on the quality of the insights which will be gained during the development of the model. For the development of a model, the developer must understand the goals of the project and select the values of the appropriate features for the training of the model. There are various techniques for improving the quality of a data set features like feature selection and featuring engineering. These techniques require extensive user experience for proper application. For the creation of the model the features which will be used must be scaled. Scale methods transform features by scaling each feature to a given range. The most common scale methods are standard and min-max scale methods. Standard scaler assumes data is normally distributed within each feature and scales them such that the distribution is centered around 0, with a standard deviation of 1. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. On the other, the min-max scaler scales and translates each feature individually such that it is in the given range on the training set, e.g., between zero and one. This scaler shrinks the data within the range of -1 to 1 if there are negative values. We can set the range like [0,1] or [0,5] or [-1,1]. Below are presented data sets that were built and used for the training of models as countermeasures against HTs.

5.3.1 Initial Data Set Development

As mentioned, 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 [141], 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 5.1). 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 thesis. 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%).

Table 5.1 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

5.4 Machine Learning Classifiers Development

The next step for the development of an ML-based model includes the selection of a suitable ML-based algorithm for the training of our model. For the development of an ML-based model, there is often more than one algorithm that can be used. The type of problem for which we aim to build our model is the most important criterion for selecting the most suitable algorithm for its development. According to this criterion, we can choose more than one algorithm which is indicated as a solution for our problem. Another criterion consists of the structure of the data set which will be used for the training and evaluation of the model. According to the features of the data set maybe we need to choose other types of algorithms. Also, it is significant to know the complexity and the speed of each algorithm, because each algorithm needs specific computing power, according to the parameters uses for the development of a model. There is a case that we cannot build our model due to lack of computer power. Must be mentioned that choosing more complex algorithms does not necessarily mean that it will achieve maximum results.

The process of training a model is the most important step of ML methodology because according to this step we produce our final ML-based model. Each training step consists of updating the weights and the biases. Training a model simply means learning/determining good values for all the weights and biases based on our data set samples. A model can be created based on labeled data samples in supervised learning and trying to leak inferences from not labeled data in unsupervised ML. For the training to be used a set of hyperparameters needs to update the weights to have better results from cycle to cycle. So, as the number of training steps grows then we can get more accurate results. However, before getting into the training process we should tweak the parameters of the model and experiment with the different results, to get the optimal ones.

To be able to develop our ML-based classifier for our REAL-880 data set we trained and optimized seven ML-based classification models. 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.

5.4.1 GB-based Classifier

GB as mentioned consists of a member of the model family and can handle features with low predictive power internally. GB models are parts of ensemble learning algorithms, which rely on a collective decision from inefficient prediction models known as decision trees. During the boosting step, each new tree is based on a modified version of the original data set. To begin, GB constructs a decision tree and assigns equal weight to each observation. Following the initial tree assessment, the weights for the easy-to-classify observations decrease while the weights for the difficult-to-classify observations grow. Then, the next tree grows on the weighted data, attempting to enhance the first tree's predictions. The new model is an amalgamation of the first and second trees. The classification error is calculated, and a third tree is built to forecast the corrected residuals. This technique is performed for a set number of iterations until convergence is reached. The final ensemble model's forecast is the weighted total of the predictions provided by all previous model iterations. The most common hyperparameters for the training of GB-based models are learning rate, number of estimators, max tree depth and max features. Number of estimators consists of the total number of sequential trees to be modeled. Max tree depth parameter controls the depth of the individual trees. And max features parameter is the number of features that will be used for the best split of the model. In Figure 5.3 is presented a typical structure of a GB algorithm.

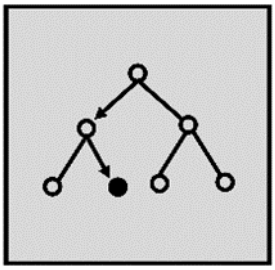


Figure 5.3 GB algorithm

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. In Table 5.2 are presented the range of the hyperparameters used and combined for the development of our GB-REAL-880 classifier. Also, in Figure 5.4 are presented the histograms with the most important features for the development of our GB-REAL-880 classifier. The "conditional total power" with "numbers of ports" and "number of cells" were the most important features. Those features helped our model to increase the classification between the two given classes.

Table 5.2 Table with the range of hyperparameters for the GB-REAL-880 classifier

Hyperparameter	Range
Learning rate	0.05 – 1
Number of estimators	10 – 100
Max tree depth	1 – 10
Max features	1 – 11

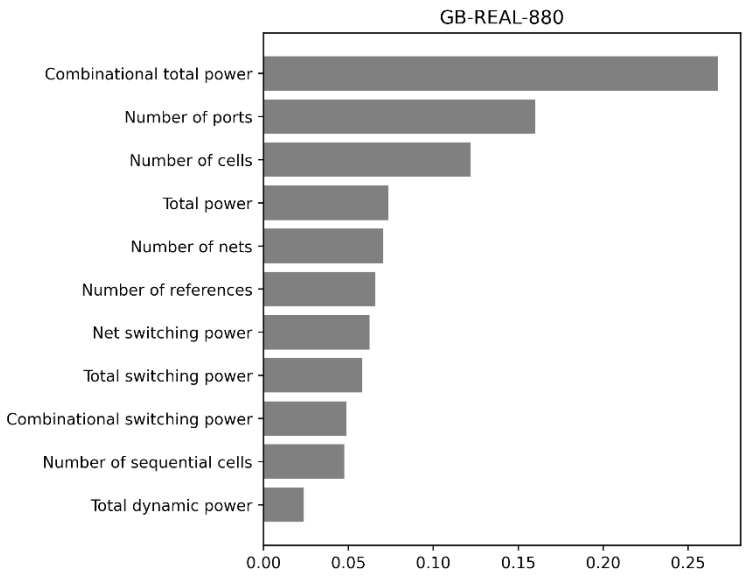


Figure 5.4 Feature importance for GB-REAL-880 classifier

5.4.2 KNN-based Classifier

KNN is a type of IB learning that can be used for solving supervised regression and classification problems simply and easily. The KNN algorithm is based on the assumption that the same things exist in a close area. In other words, similar things are close to one

another. KNN is based on the idea of similarity (also known as distance, proximity, or closeness) figuring the space between points on a graph. There are various methods of calculating distance, and one way might be preferable depending on the problem. The KNN algorithm is initially loaded with the training data set, which is commonly referred to as x, and their goal values, which are referred to as y. Goal value y needs to be classified. Then is initialized k to a preferable number of neighbors and for each data sample is computed the distance between the sample whose target value is wanted to classify. Next, are added both the index and the distance of the query example to an ordered list of indices and distances and sort this list in ascending order (from smaller to bigger), with the distance as order criteria. Finally, are picked the first k entries from the sorted list are got the labels of the selected k entries. So can be returned in the form of the k labels. Some of the most often used hyperparameters for the training of a KNN-based model are a number of neighbors, leaf size and weights metrics. A number of neighbors are used to returned indices of and distances to the neighbors of each point. Leaf size parameter that is to say the maximum number of points a node can hold. Weights parameter is used to approximate the optimal degree of influence of individual features using a training set. When successfully applied relevant features are attributed a high weight value, whereas irrelevant features are given a weight value close to zero. In Figure 5.5 is presented a typical structure of a KNN algorithm.

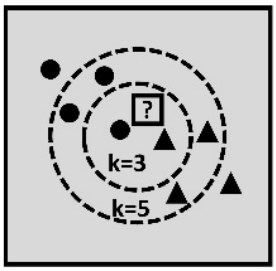


Figure 5.5 KNN algorithm

For the development of our KNN-based classifier, we used five hyperparameters: number of neighbors, distances, leaf size and weights. 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 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 and weights 'uniform'. In Table 5.3 are presented the range of hyperparameters used and combined for the development of our KNN-REAL-880 classifier for the real data set.

Table 5.3 Table with the range of hyperparameters for the KNN-REAL-880 classifier

Hyperparameter	Range
Number of neighbors	1 – 60
Distances	1 – 10
Leaf size	1 – 50
Weights	uniform, distance

5.4.3 LR-based Classifier

LR is a supervised ML algorithm used for classification problems, and specifically for categorizing observations into a group of discrete classes. Although linear regression assigns observations to a continuous number of values, LR applies on its output a transformation - activation – function, called the logistic sigmoid function. It returns a probability value which can then be matched with two or more classes. LR is used when the target – dependent - variable is categorical. For example, to predict whether an email is a spam (1) or not (0) (binary LR) or to predict whether a car with specific characteristics belongs to a model, etc. (multiclass LR). In Figure 5.6 is presented a typical structure of an LR algorithm.

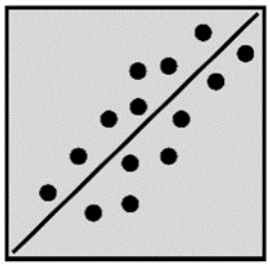


Figure 5.6 LR algorithm

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 [142], limited-memory Broyden–Fletcher–Goldfarb–

Shanno (LM-BFGS) [143], library large-scale linear LIBLINEAR [144], stochastic average gradient (SAG) [145] and SAGA [146] 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. In Table 5.4 are presented the range of hyperparameters used and combined for the development of our LR-REAL-880 classifier for the real data set.

Table 5.4 Table with the range of hyperparameters for the LR-REAL-880 classifier

Hyperparameter	Range
Solver	newton-cg, lm-bfgs, liblinear, sag, saga
Penalty	l1, l2, elasticnet
C	0.01 – 1000
Max iterations	100 – 7000

5.4.4 MLP-based Classifier

ANNs are built as the model of neurons present in the human brain. Based on the philosophy of ANNs the algorithm MLP consists of a feedforward ANN that generates a set of outputs from a set of inputs. Specifically, an MLP is a neural network that connects multiple layers in a directed graph, meaning that the signal route across the nodes is only one direction. Aside from the input nodes, each node has a nonlinear activation function. MLP is frequently utilized for supervised learning tasks. Common hyperparameters for an MLP model are hidden layer sizes, activation, solver, alpha, max iterations and learning rate. Hidden layers size is used for the creation of the hidden layers. The hidden layers are produced according to the size value. Also, the hidden layer simply produces layers of mathematical functions each designed to produce an output specific to an intended result. Activation hyperparameter consists of an activation function that defines how the weighted sum of the input is turned into an output from a node or nodes in a network layer. Solver parameter represents a stochastic gradient descent-based optimizer for optimizing the parameters in the computation graph. The alpha parameter is a regularization term, also known as a penalty term, that combats overfitting by limiting the size of the weights. Increasing alpha may

alleviate high variance by encouraging smaller weights, resulting in a decision boundary plot with fewer curvatures. An iteration is the number of times a batch of data is processed by the algorithm. In the context of neural networks, this refers to the forward and backward passes. As a result, each time you run a batch of data through the ANN, you complete an iteration. The learning rate, in particular, is an adjustable hyperparameter used in neural network training that has a tiny positive value, typically in the range of 0.0 to 1.0. The learning rate determines how quickly the model adapts to a new situation. It could be the model's most essential hyperparameter. In Figure 5.7 is presented a typical structure of an MLP algorithm.

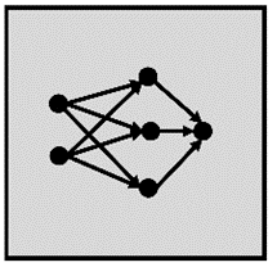


Figure 5.7 MLP algorithm

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’. In Table 5.5 are

presented the range of hyperparameters used and combined for the development of our MLP-REAL-880 classifier for the real data set.

Table 5.5 Table with the range of hyperparameters for the MLP-REAL-880 classifier

Hyperparameter	Range
Hidden layer sizes	10 – 50
Activation	identity, logistic, tanh, relu
Solver	lm-bfgs, sgd, adam
Alpha	0.001 – 0.9
Max iterations	100 – 1000
Learning rate	constant, adaptive, invscaling

5.4.5 RF-based Classifier

RF consists of a summation of Decision Trees. The general idea of this technique is that a mixture of learning models raises the general result. RF builds multiple decision trees and merges them together to achieve the preciseness and stability of the prediction. In that way, it prevents overfitting by creating random subsets of the features, building smaller trees using these subsets and combining them to increase the overall performance. RF categorizes a sample to the class with the maximum "votes" among each subtree. RF makes the model more random while developing the trees. Rather than looking for the most significant feature while splitting a node, it scans for the best element among a random subset of features. This outcome in a wide variety that by and large results in a greater model. Some of the most common hyperparameters for the training of an RF-based model are a number of estimators, max features, max depth and min sample leaf. Number of estimators is the number of trees that are used to construct before calculating the maximum voting or prediction averages. A greater number of trees improves performance but needs more computer power. Max features parameter is used to determine the maximum number of features RF is allowed to try an individual tree. For instance, if the total number of variables is 100, we can only take 10 of them in the individual tree. Max depth parameter represents the depth of each tree in the forest The deeper the tree, the more splits it has and the more information it captures about the data. Min sample leaf parameter represents the minimum number of samples required to be at a leaf node. In Figure 5.8 is presented a typical structure of an RF algorithm.

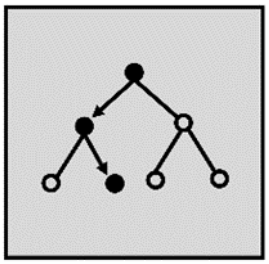


Figure 5.8 RF algorithm

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. In Table 5.6 are presented the range of hyperparameters used and combined for the development of our RF-REAL-880 classifier for the real data set.

Table 5.6 Table with the range of hyperparameters for the RF-REAL-880 classifier

Hyperparameter	**Range**
Number of estimators	100 –5000
Max features	auto, sqrt, log2
Max depth	2 – 50
Min sample leaf	1 – 20

5.4.6 SVM-based Classifier

SVM is an algorithm intrinsically for binary problems. SVMs transform the input feature space into higher-dimensional feature space using the kernel trick dot product. Each data set's sample distance can be found to a given dividing hyperplane. Margin is called the minimum distance from the samples to the hyperplane. The transformed data can be separated using a hyperplane, the dividing curve between distinct classes. The optimal hyperplane maximizes the margin. Its goal is to classify a new sample by simply computing

the distance from the hyperplane. Based on global optimization, SVMs deal with overfitting problems, which appear in high-dimensional spaces, making them appealing in various applications [147][148]. Most used SVM algorithms include the support vector regression [149], least squares SVM [150] and successive projection algorithm-SVM [151]. In other words, an SVM is a linear separator that focuses on creating a hyperplane with the largest possible margin. Its goal is to classify a new sample by simply computing the distance from the hyperplane. On a two-dimensional feature space, the hyperplane is a single line dividing the two classes. On a multi-dimensional feature space, where the data are non-linearly separable an SVM cannot linearly classify the data. In this case, it uses the kernel trick. The main concept has to do with the fact that the new multidimensional feature space could have a linear decision boundary which might not be linear in the original feature space. Common in use SVM hyperparameters is C, gamma and kernel. The C parameter instructs the SVM optimizer how much you wish to avoid misclassifying each training example. For large values of C, the optimization will select a smaller-margin hyperplane if it does a better job of accurately classifying all of the training points. The gamma parameter defines how far a single training example's impact extends, with low values indicating 'far' and large values indicating 'near.' The gamma parameters can be thought of as the inverse of the radius of influence of samples chosen as support vectors by the model. A kernel function is a way for taking data as input and transforming it into the needed form for processing. The term "kernel" is chosen because the collection of mathematical functions utilized in SVM provides a window through which data can be manipulated. In Figure 5.9 is presented a typical structure of an SVM algorithm.

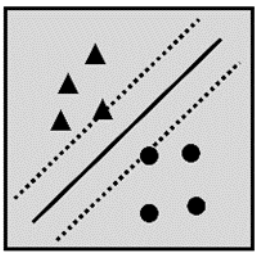


Figure 5.9 SVM algorithm

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'. In Table 5.7 are presented the range of hyperparameters which used and combined for the development of our SVM-REAL-880 classifier for the real data set.

Table 5.7 Table with the range of hyperparameters for the SVM-REAL-880 classifier

Hyperparameter	Range
C	0.0001 –100
Gamma	scale, auto
Kernel	linear, polynomial, gaussian radial basis

5.4.7 XGB-based Classifier

XGB belongs to the family of ensemble learning methods. Sometimes, it could be insufficient to depend on the results of only one ML method applied to our data. Ensemble learning techniques use a systematic method to combine the predictive power of various learning methods. The output of this combination is a model that provides the totaled result from smaller-weaker- models. Most of the time, we use the XGB algorithm with decision trees.

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. In Table 5.8 are presented the range of hyperparameters which used and combined for the development of our XGB-REAL-880 classifier for the real data set.

Table 5.8 Table with the range of hyperparameters for the XGB-REAL-880 classifier

Hyperparameter	Range
Learning rate	0.05 – 1
Number of estimators	10 – 100
Max tree depth	1 – 11

5.5 Machine Learning Classifiers Evaluation

Once we have completed the steps of data collection and preparation, and after we select algorithms and train our model, it is time to evaluate our model. For the evaluation of our model is used a test set which mainly consisted of 20% of the total data set and the samples of this set are unknown to our model. For example, in the case of HTs classification, the test set consisted of unknown infected and free circuits features which the model will process for the first time and needs to classify.

In this thesis, 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-Sensitivity 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).

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

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

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

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

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

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

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

From Figure 5.10, 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 5.11). 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.

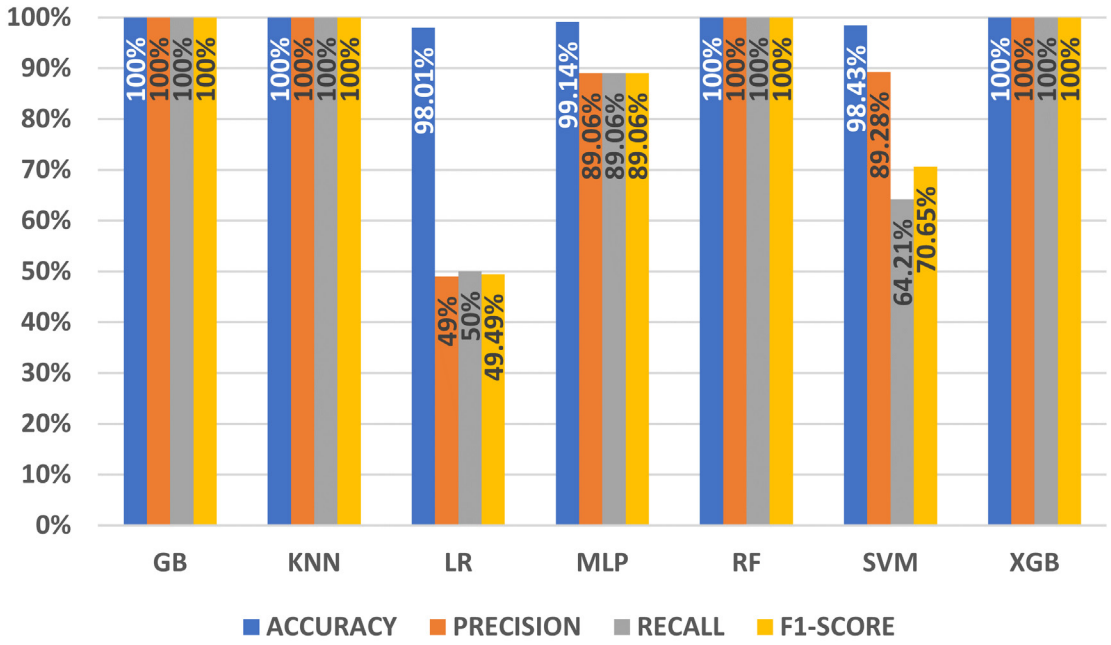


Figure 5.10 Histograms of the performance of our seven ML models on our REAL-880 training set

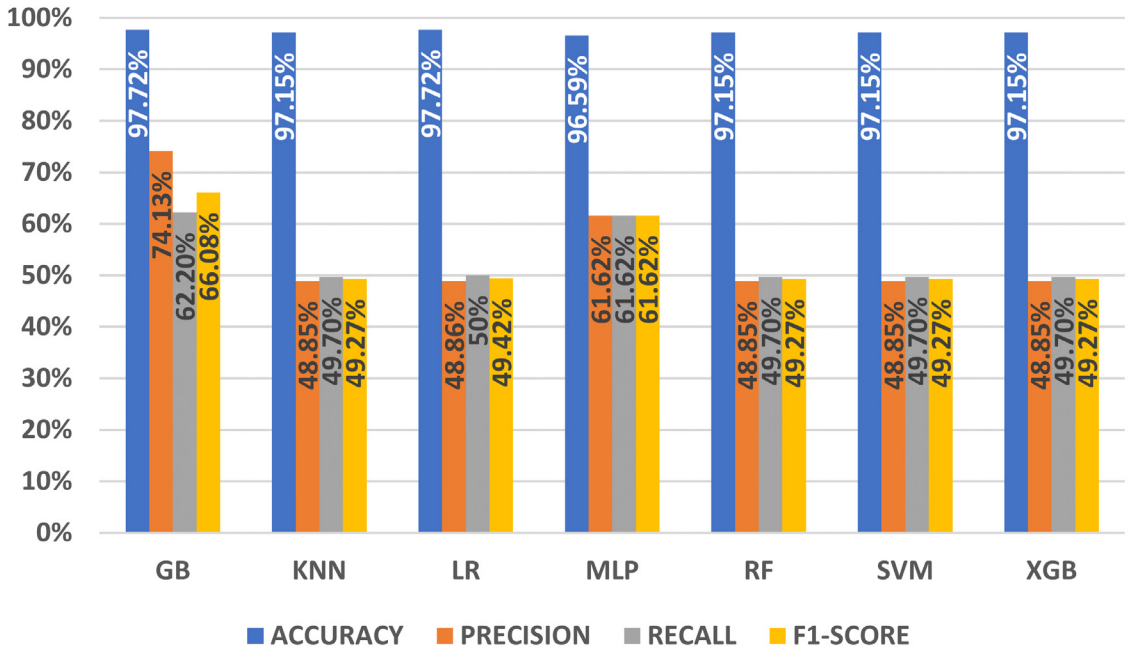


Figure 5.11 Histograms of the performance of our seven ML models on our REAL-880 test set

5.6 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 5.12).

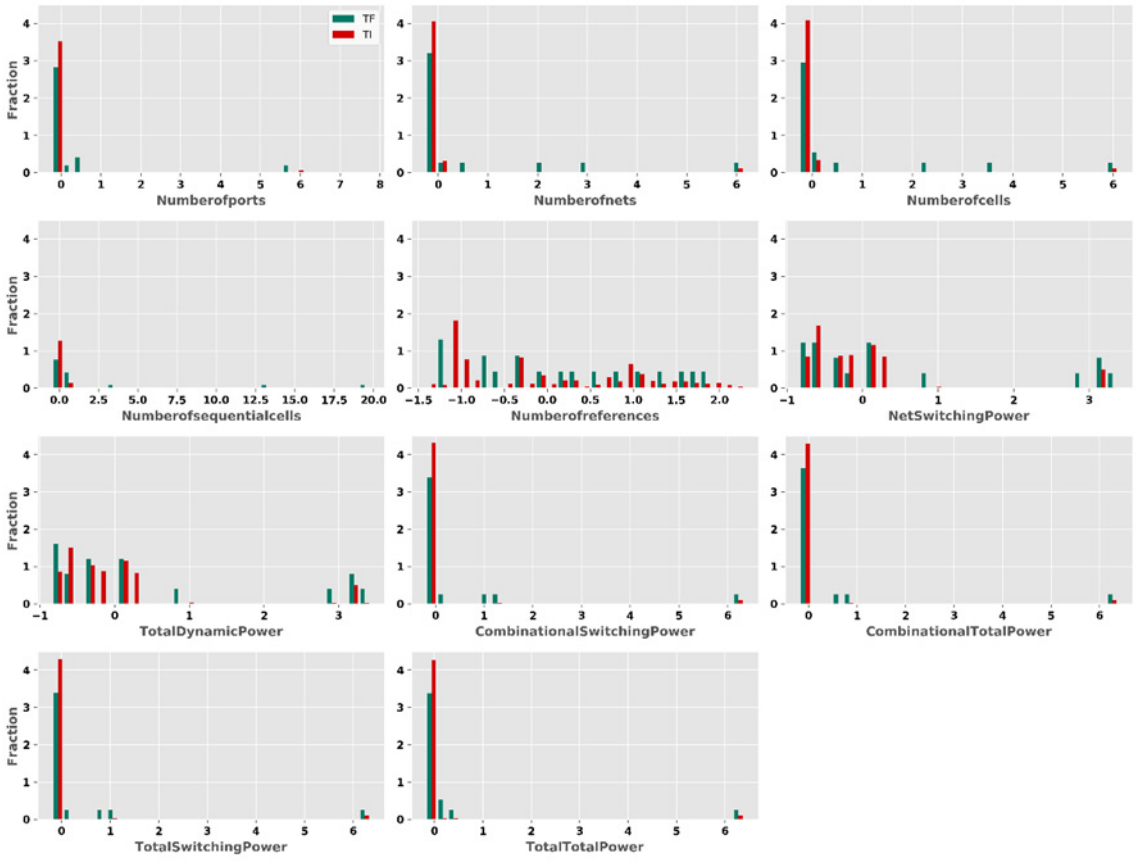


Figure 5.12 Data distributions by feature and class

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 a 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 thesis, we modified the networks to 1D (one-dimensional) networks based on DNNs, because our data set consisted of 1D features.

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.

5.6.1 GAN, CGAN, WGAN & WCGAN Algorithms

GL algorithms aim to generate new synthetic samples and they can be applied as a solution for the imbalanced data sets. In this section is mentioned GL-based algorithms which can be used for the synthesis of new samples for databases cases such as normal and infected circuits. For the development of GL-based models must be developed as many models as the number of classes that are contained in the data set. Then, depending on the algorithm which will be used, there may be a need to applied some clustering algorithms. With the use of the clustering algorithms the user will be able to cluster each given class to sub-classes in order to be able to use the class label as an extra feature.

Specifically, GANs consist 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 a 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) [152], Adam [153] and root mean square propagation (RMSprop) [154]. 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) [155], sigmoid [156] and hyperbolic tangent (Tanh) [157] (Table 5.9).

Table 5.9 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 Table 5.10 and Table 5.11 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 Table 5.12 and Table 5.13 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 5.10 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 5.11 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 5.12 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 5.13 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

5.7 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 5.13) and discriminator loss value equal to 0.0984 (Figure 5.14). 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 5.13) and discriminator loss value equal to 0.114 (Figure 5.14), 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 5.13) and discriminator loss value equal to 0.263 (Figure 5.14). 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 5.13) and discriminator loss equal to 0.273 (Figure 5.14).

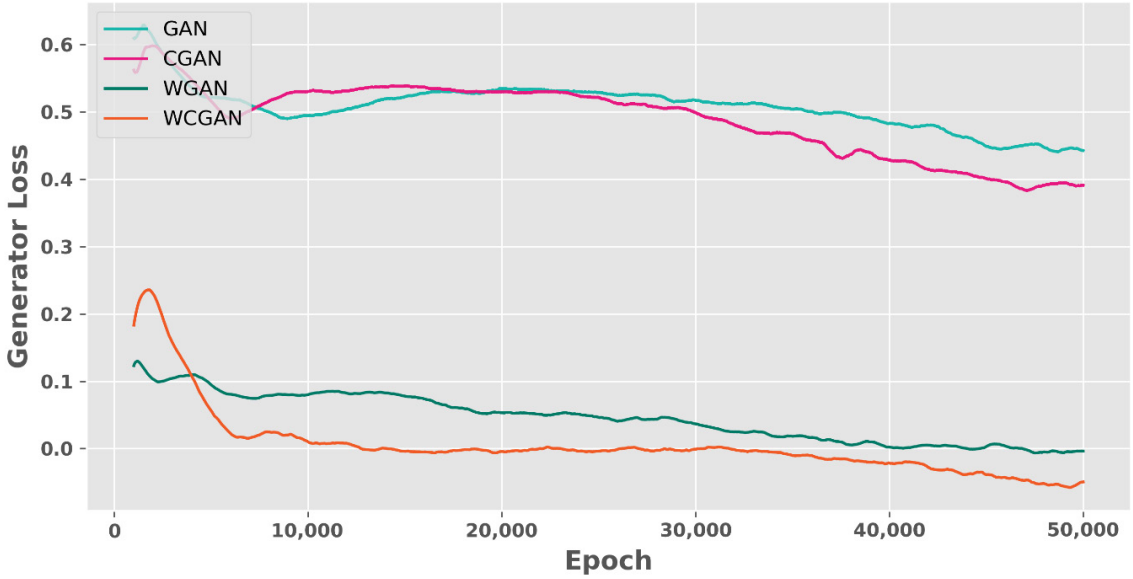


Figure 5.13 Generator loss values of our four models for each epoch

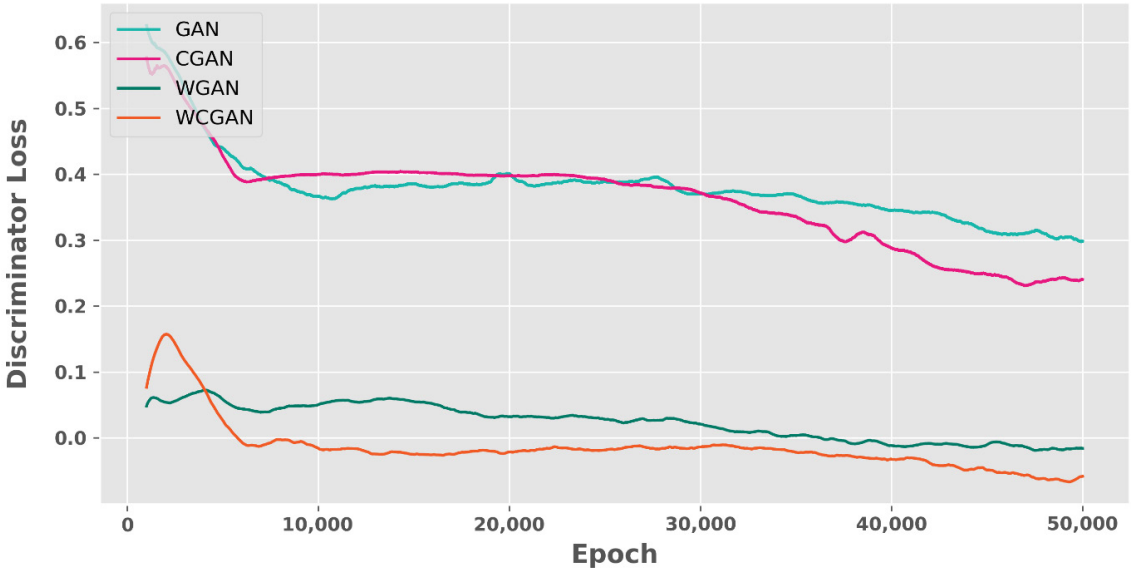


Figure 5.14 Discriminator loss values of our four models for each epoch

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 5.15) was able to synthesize better-generated samples compared with the other models and especially compared with our worst-performing GAN-based model (Figure 5.16). 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.

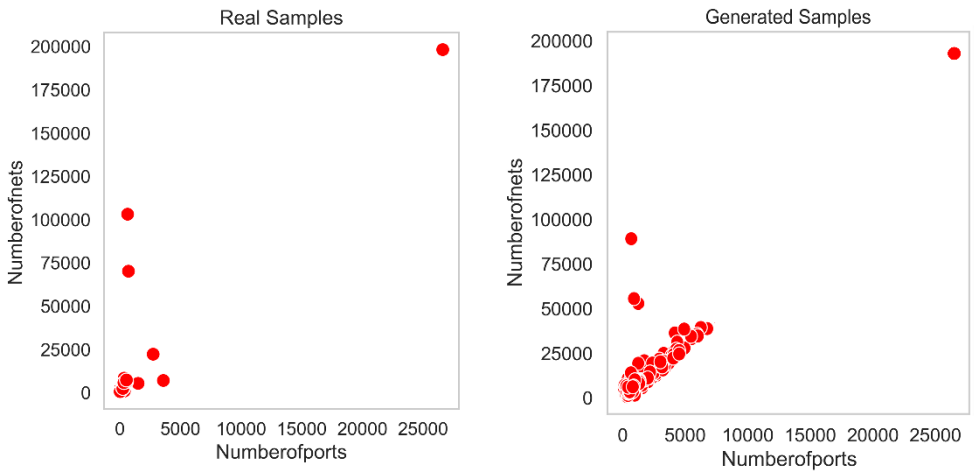


Figure 5.15 Presentation of how our best-performing WCGAN-based model learned to synthesize new generated samples based on real samples

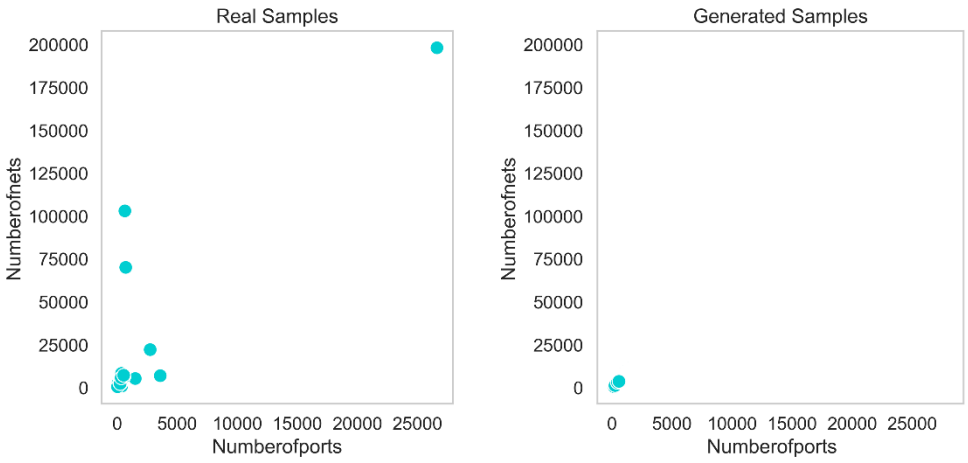


Figure 5.16 Presentation of how our worst-performing GAN-based model learned to synthesize new generated samples based on real samples

5.8 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, were 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 5.17).

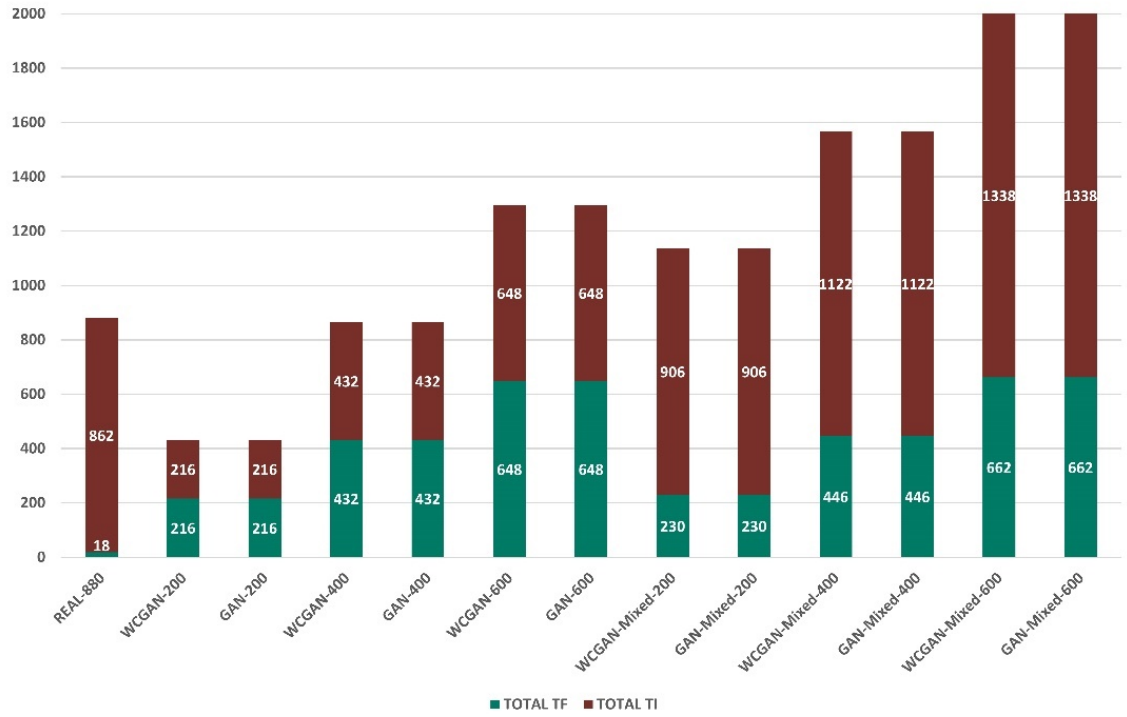


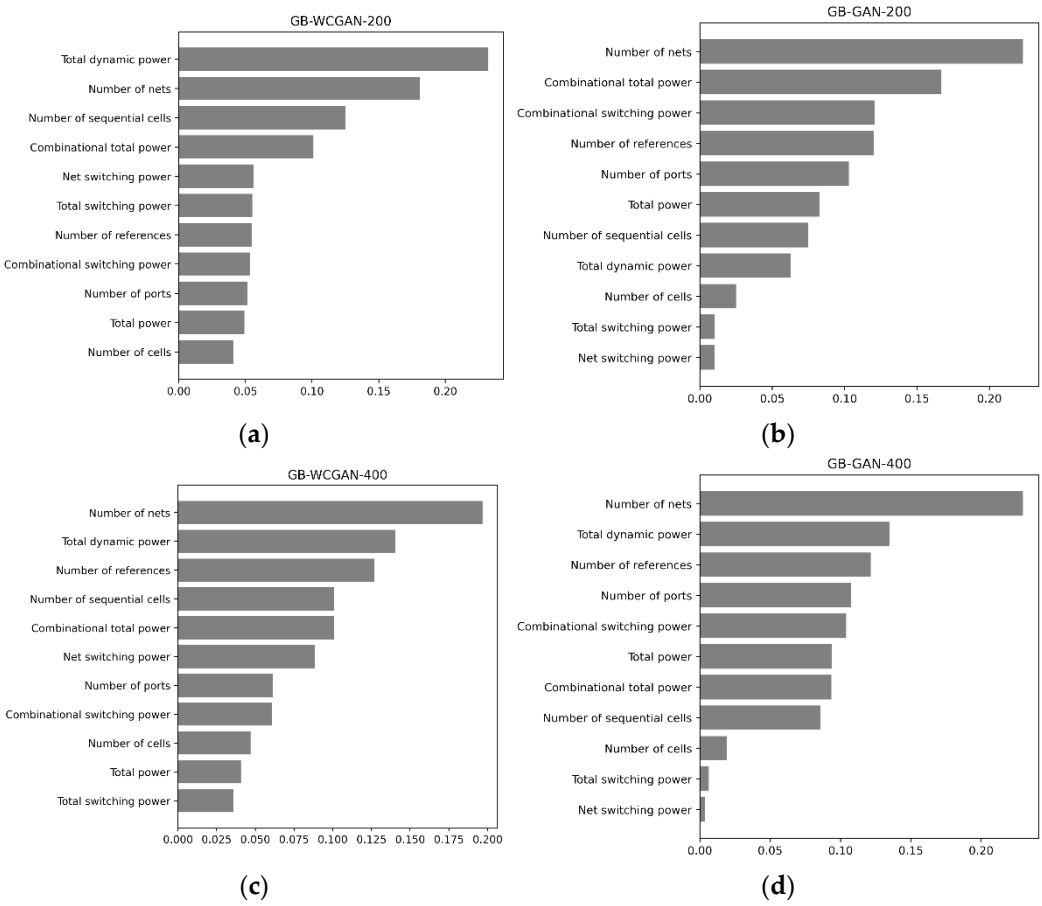
Figure 5.17 Histograms with the distribution of TF and TI samples for our 13 data sets

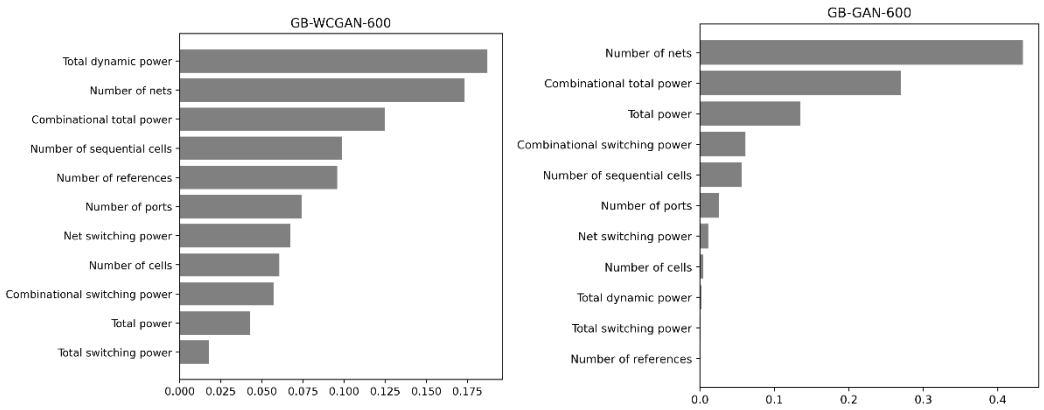
5.9 New Generated GB-based Classifiers Development

As mentioned, for the classification of our REAL-880 data set the best ML-based classifier from the seven compared algorithms was the GB-based classifier. As a result, we based on GB-algorithm for the classification of our new generated data sets. As previously for the development of our new generated GB-based classifiers we used and combined a list of four hyperparameters: learning rate, max tree depth, number of estimators and max features. Specifically, we used a list of learning rate values from 0.05 to 1, a list of number of estimators with values from 10 to 100, a list of max tree depth values from 1 to 10 and a list of max features values from 1 to 11. In Table 5.14 are presented the best combination of hyperparameters for each of the six new generated GB-based classifiers. Also, in Figure 5.18 are presented the most important features for each new generated GB-based classifier. It is observed that the most important feature for the six classifiers was “number of nets”. While for the GB-WCGAN-based classifiers the next most important feature was the “total dynamic power” and for the GB-GAN-based was the “combinational total power”.

Table 5.14 Table with the range of hyperparameters for the new generated GB-based classifiers

Classifier	Learning rate	Number of estimators	Max tree depth	Max features
GB-WCGAN-200	0.05	10	1	6
GB-GAN-200	0.05	10	1	3
GB-WCGAN-400	0.05	10	1	6
GB-GAN-400	0.05	10	1	3
GB-WCGAN-600	0.05	10	1	10
GB-GAN-600	0.05	10	2	3





(**e**) (**f**)

Figure 5.18 Concept graph presenting the most importance features: (a) GB-WCGAN-200 classifier; (b) GB-GAN-200 classifier; (c) GB-WCGAN-400 classifier; (d) GB-GAN-400 classifier; (e) GB-WCGAN-600 classifier; (f) GB-GAN-600 classifier

5.10 Mixed GB-based Classifiers Development

Our next step, was the development of mixed GB-based classifiers for the classification of our mixed data sets. Again, for the development of our mixed GB-based classifiers we used and combined a list of four hyperparameters: learning rate, max tree depth, number of estimators and max features. Specifically, we used a list of learning rate values from 0.05 to 1, a list of number of estimators with values from 10 to 100, a list of max tree depth values from 1 to 10 and a list of max features values from 1 to 11. In Table 5.15 are presented the best combination of hyperparameters for each of the six mixed GB-based classifiers. Also, in Figure 5.19 are presented the most important features for each mixed GB-based classifier. It is observed that the most important feature for the six classifiers was “number of sequential cells”. While for the GB-WCGAN-Mixed-based classifiers the next most important feature was the “combinational total power” and for the GB-GAN-Mixed-based was the “number of ports”.

Table 5.15 Table with the best values of hyperparameters for the mixed GB-based classifiers

Classifier	Learning rate	Number of estimators	Max tree depth	Max features
GB-WCGAN-Mixed-200	0.75	40	10	4
GB-GAN-Mixed-200	1	20	9	3
GB-WCGAN-Mixed-400	0.05	50	2	9

GB-GAN-Mixed-400	0.05	20	4	8
GB-WCGAN-Mixed-600	0.05	20	6	7
GB-GAN-Mixed-600	0.05	10	5	7

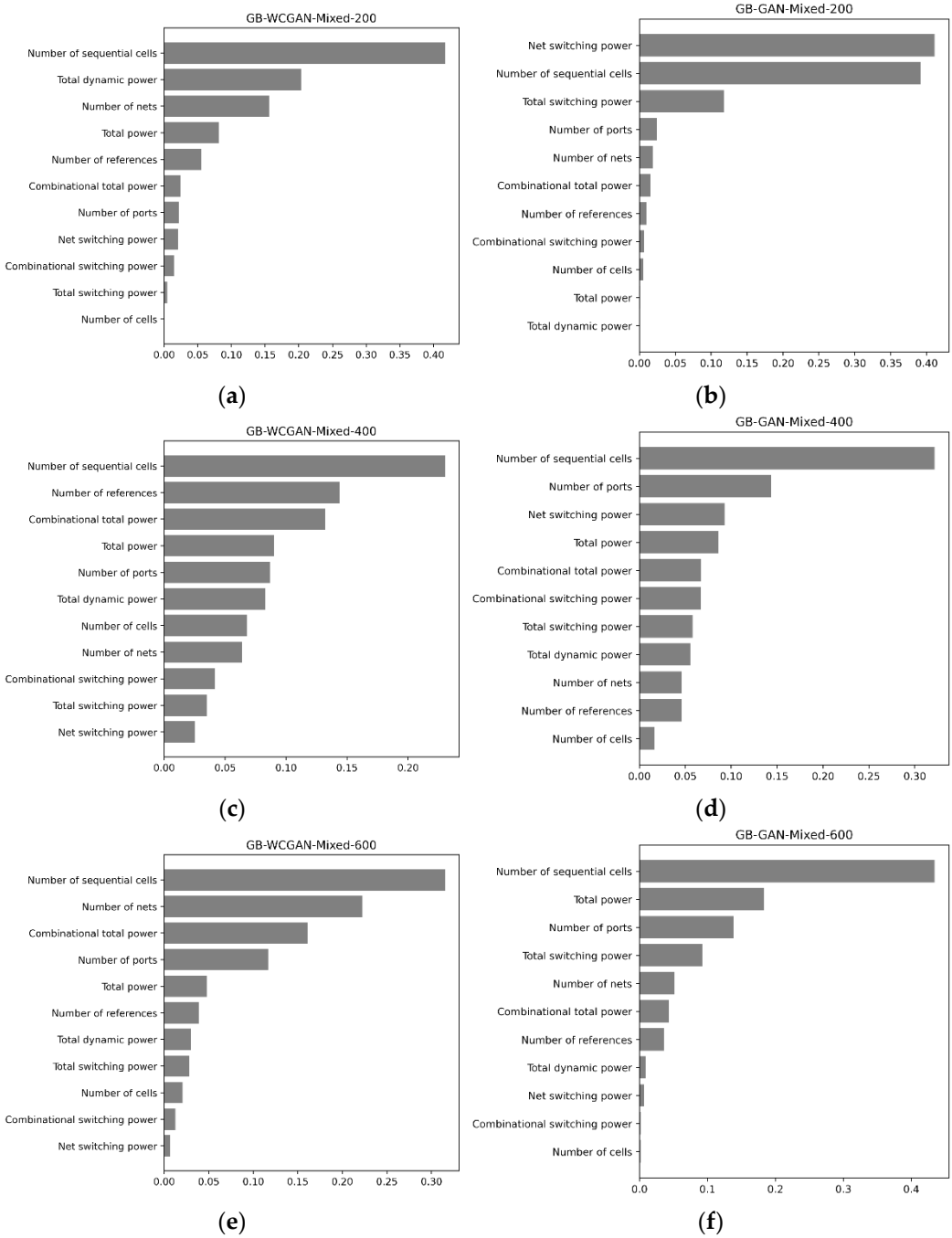


Figure 5.19 Concept graph presenting the most importance features: (a) GB-WCGAN-Mixed-200 classifier; (b) GB-GAN-Mixed-200 classifier; (c) GB-WCGAN-Mixed-400 classifier; (d) GB-GAN-Mixed-400 classifier; (e) GB-WCGAN-Mixed-600 classifier; (f) GB-GAN-Mixed-600 classifier

Chapter 6 Results

6.1 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 Figure 6.1 and Figure 6.2, 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.

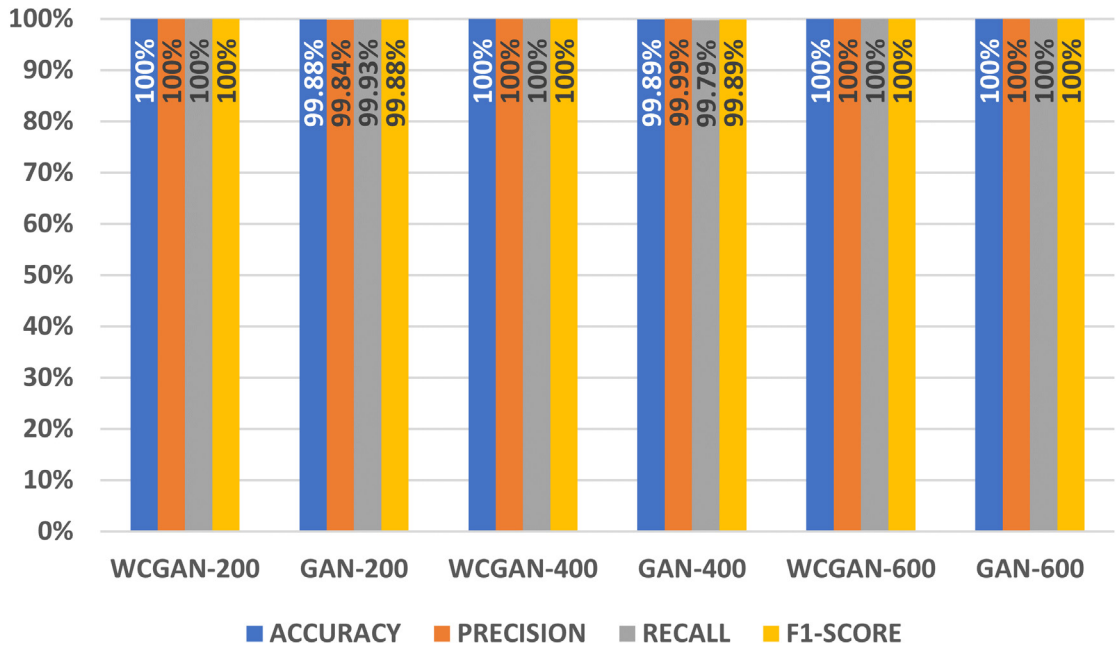


Figure 6.1 Histograms of the performance of our new GB-based classifiers on our new generated training sets.

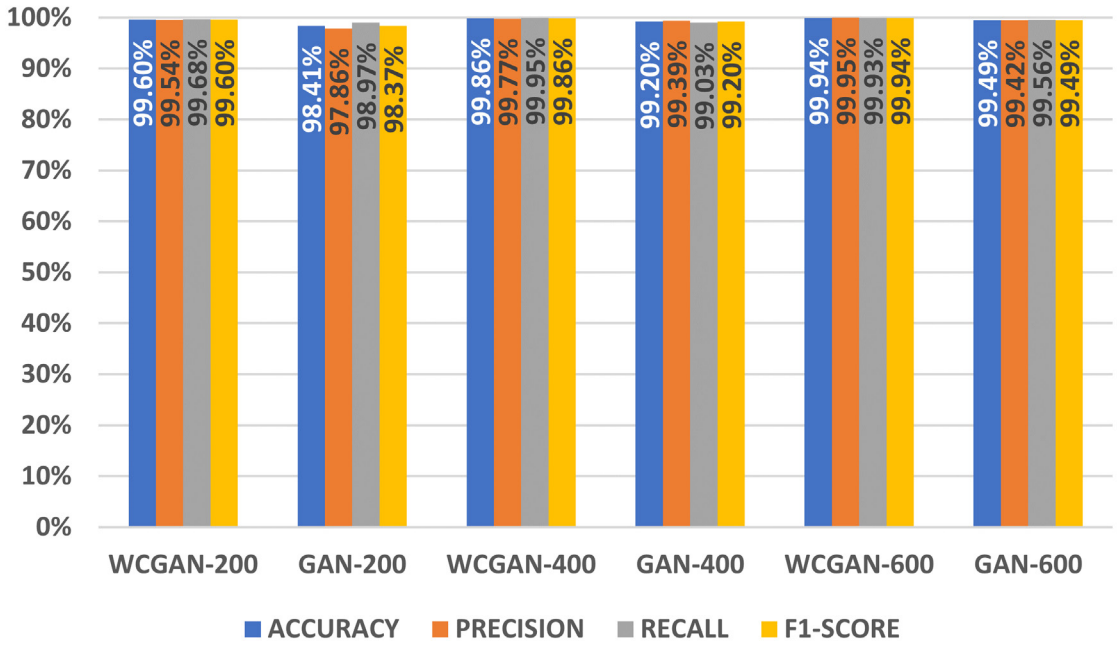


Figure 6.2 Histograms of the performance of our new GB-based classifiers on our new generated test sets.

6.2 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 Figure 6.3 and Figure 6.4 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.

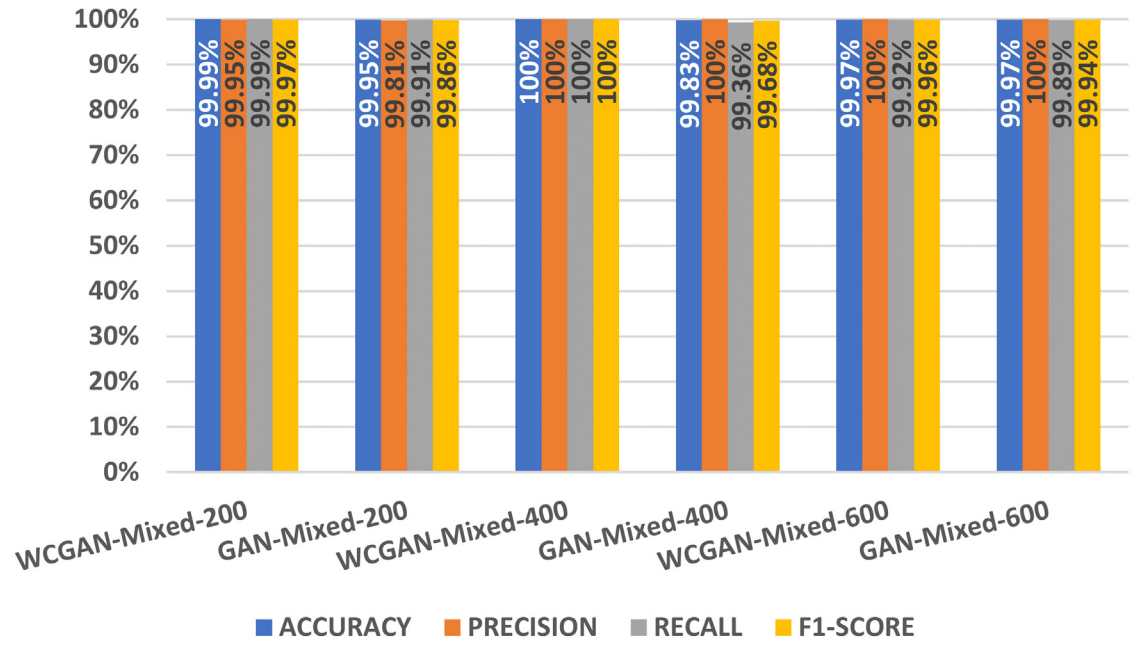


Figure 6.3 Histograms of the performance of our new GB-based classifiers on our mixed training sets.

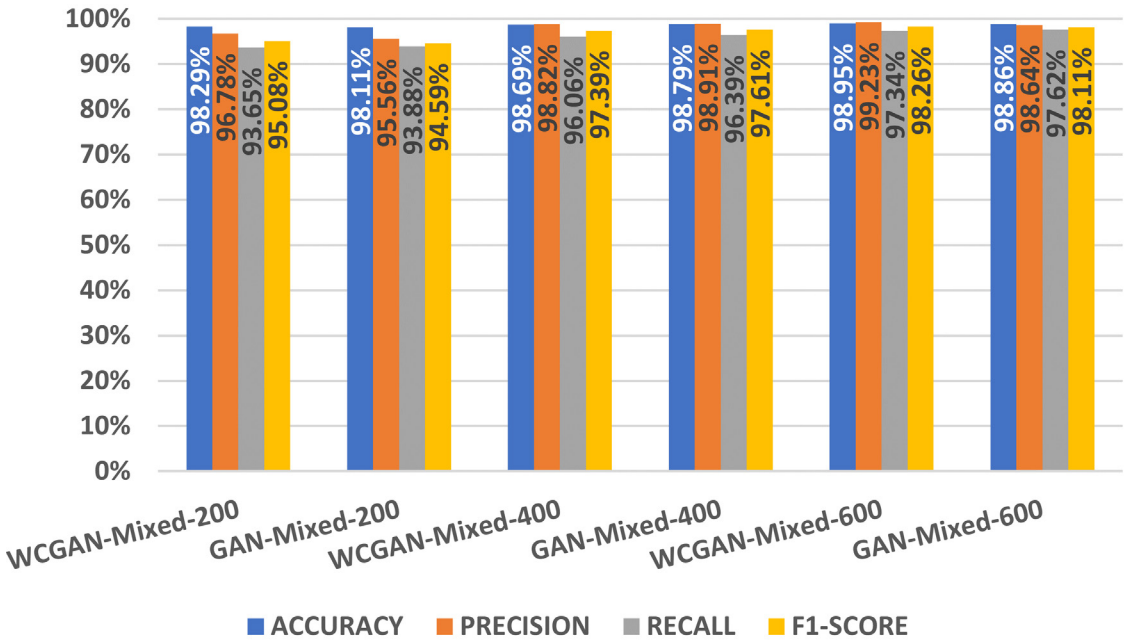


Figure 6.4 Histograms of the performance of our new GB-based classifiers on our mixed test sets.

6.3 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 6.5).

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 6.6, it can be observed that our WCGAN-600 (Figure 6.6 c,d) and WCGAN-Mixed-600 (Figure 6.6 g,h) data sets significantly enhanced the classification procedure compared with our GAN-600 (Figure 6.6 e,f) and GAN-Mixed-600 data sets (Figure 6.6 i,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.

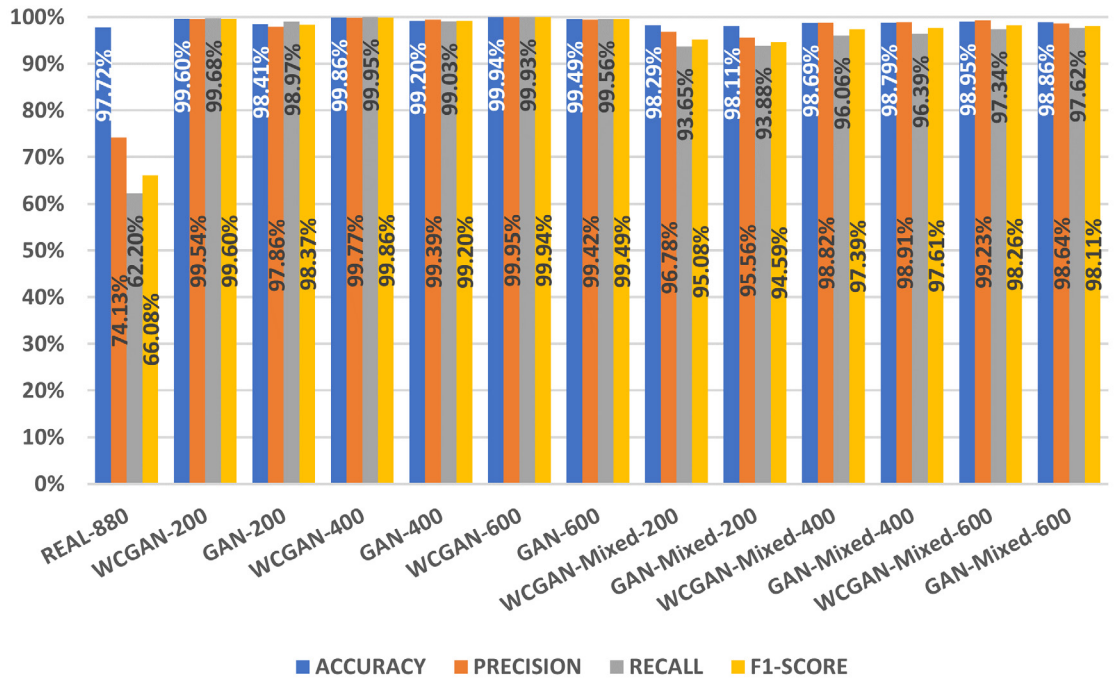
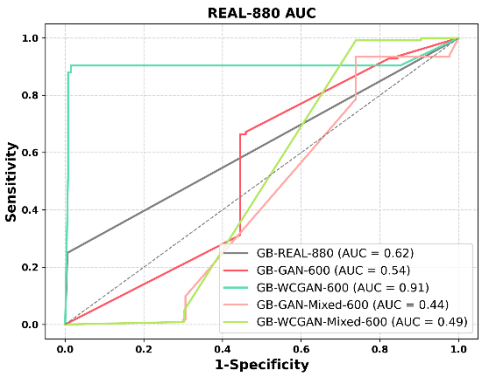
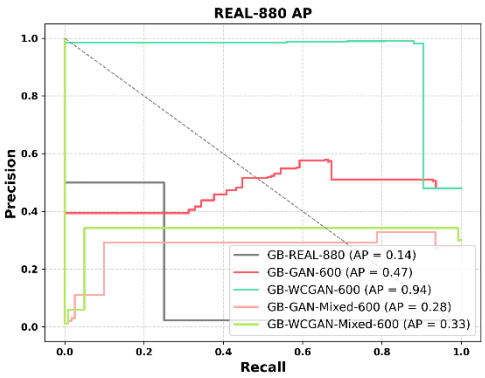


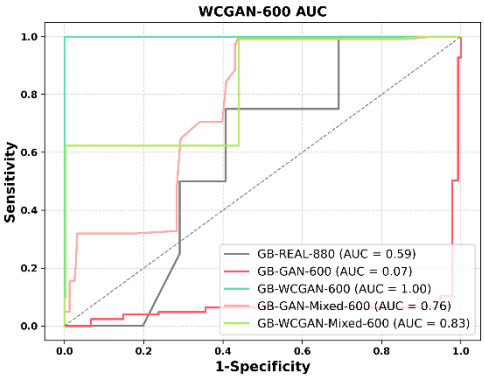
Figure 6.5 Histograms of the performance of our 13 GB-based classifiers on our 13 test sets.


REAL-880 AUC
Sensitivity
1-Specificity
GB-REAL-880 (AUC = 0.62)
GB-GAN-600 (AUC = 0.54)
GB-WCGAN-600 (AUC = 0.91)
GB-GAN-Mixed-600 (AUC = 0.44)
GB-WCGAN-Mixed-600 (AUC = 0.49)

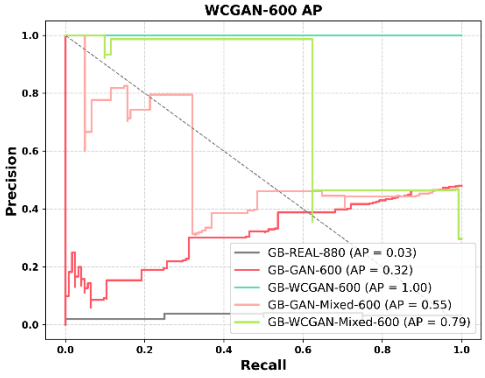
(a)


REAL-880 AP
Precision
Recall
GB-REAL-880 (AP = 0.14)
GB-GAN-600 (AP = 0.47)
GB-WCGAN-600 (AP = 0.94)
GB-GAN-Mixed-600 (AP = 0.28)
GB-WCGAN-Mixed-600 (AP = 0.33)

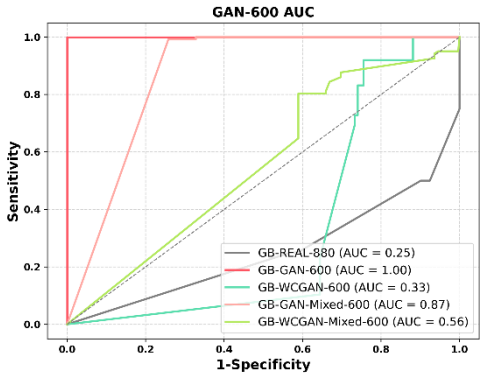
(b)


WCGAN-600 AUC
Sensitivity
1-Specificity
GB-REAL-880 (AUC = 0.59)
GB-GAN-600 (AUC = 0.07)
GB-WCGAN-600 (AUC = 1.00)
GB-GAN-Mixed-600 (AUC = 0.76)
GB-WCGAN-Mixed-600 (AUC = 0.83)

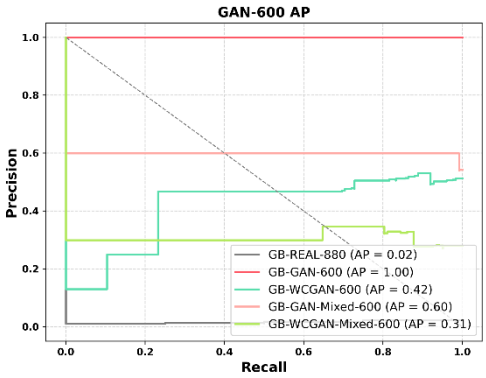
(c)


WCGAN-600 AP
Precision
Recall
GB-REAL-880 (AP = 0.03)
GB-GAN-600 (AP = 0.32)
GB-WCGAN-600 (AP = 1.00)
GB-GAN-Mixed-600 (AP = 0.55)
GB-WCGAN-Mixed-600 (AP = 0.79)

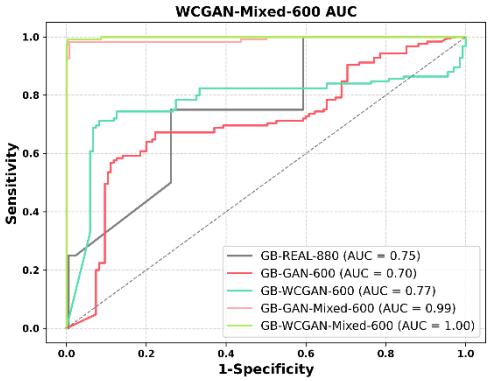
(d)


GAN-600 AUC
Sensitivity
1-Specificity
GB-REAL-880 (AUC = 0.25)
GB-GAN-600 (AUC = 1.00)
GB-WCGAN-600 (AUC = 0.33)
GB-GAN-Mixed-600 (AUC = 0.87)
GB-WCGAN-Mixed-600 (AUC = 0.56)

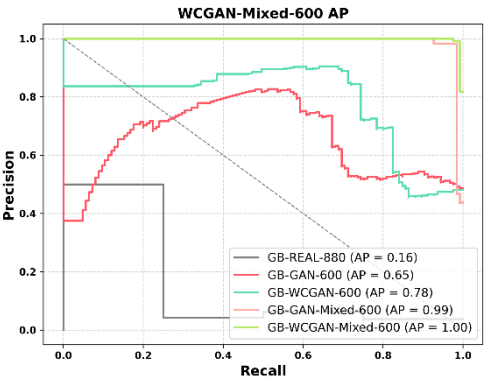
(e)


GAN-600 AP
Precision
Recall
GB-REAL-880 (AP = 0.02)
GB-GAN-600 (AP = 1.00)
GB-WCGAN-600 (AP = 0.42)
GB-GAN-Mixed-600 (AP = 0.60)
GB-WCGAN-Mixed-600 (AP = 0.31)

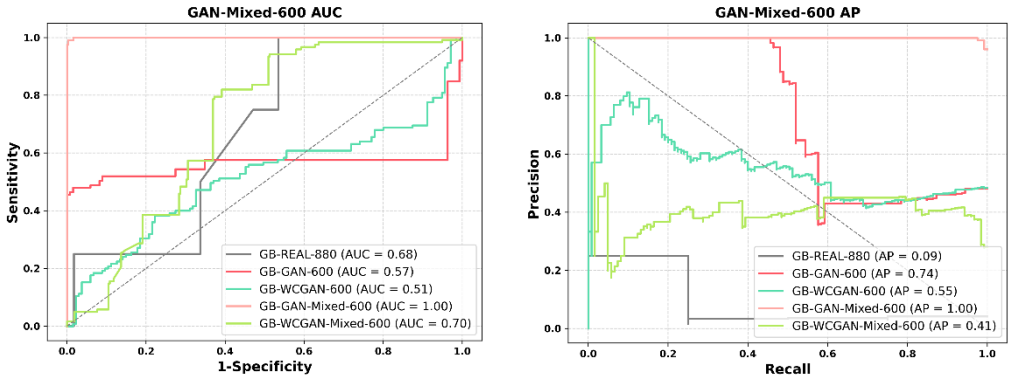
(f)


WCGAN-Mixed-600 AUC
Sensitivity
1-Specificity
GB-REAL-880 (AUC = 0.75)
GB-GAN-600 (AUC = 0.70)
GB-WCGAN-600 (AUC = 0.77)
GB-GAN-Mixed-600 (AUC = 0.99)
GB-WCGAN-Mixed-600 (AUC = 1.00)

(g)


WCGAN-Mixed-600 AP
Precision
Recall
GB-REAL-880 (AP = 0.16)
GB-GAN-600 (AP = 0.65)
GB-WCGAN-600 (AP = 0.78)
GB-GAN-Mixed-600 (AP = 0.99)
GB-WCGAN-Mixed-600 (AP = 1.00)

(h)



(i) (j)

Figure 6.6 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

6.4 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 6.7, 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 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.

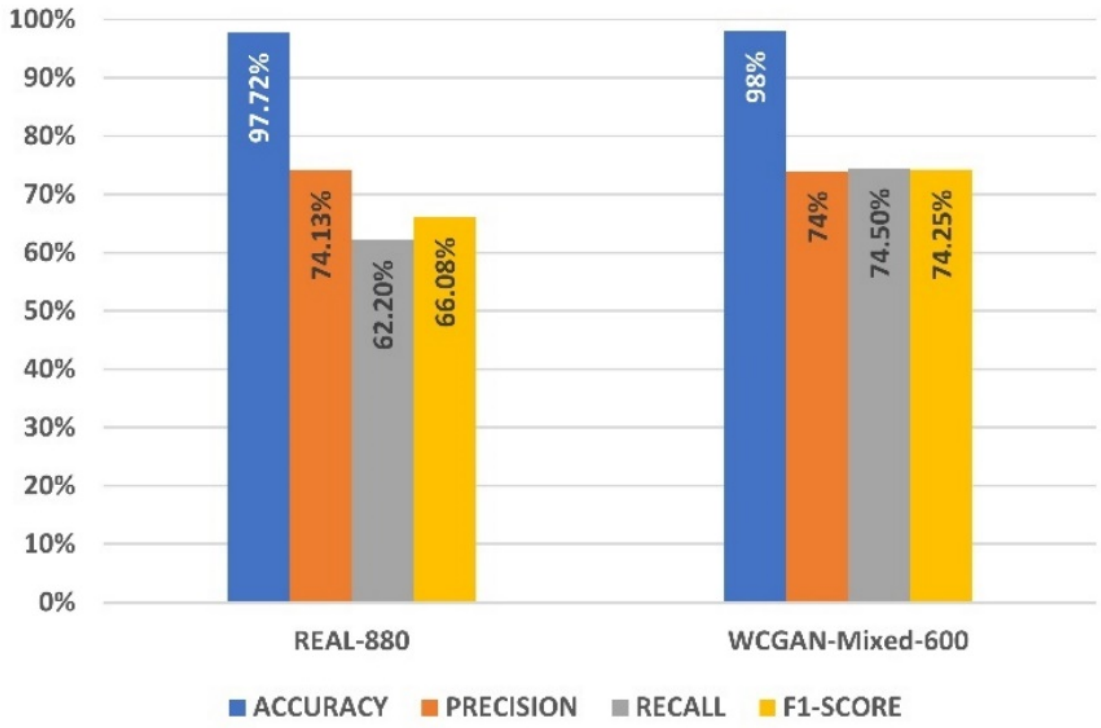


Figure 6.7 Histograms of the performance of our new best-performing GB-WCGAN-Mixed-600 classifier compared with our GB-REAL-880 classifier on the REAL-880 test set.

6.5 Comparison to Existing Methods

Our final step was to compare our best performing GB-WCGAN-Mixed-600 classifier with existing methods. As be mentioned we named GB-WCGAN-Mixed-600 classifier as ATLAS. So, we compared our ATLAS with two studies that can be found in the literature that is based on SVM [24] and RF classifiers [25].

We chose 15 circuits that existing methods were tested on, to make the comparison with our model fair Figure 6.8 and Table 6.1. Our ATLAS model exhibits the highest performance compared to existing methods, with an average Precision and F1-score of 100%.

It is worth mentioning that our ATLAS classifier for HT classification is based on area and power feature values that are extracted from the whole circuit. Therefore, we provide a prediction for the entire circuit, labeling it as TF or TI. Both existing studies however, break each circuit down to the level of nets. Each net is treated as an individual sample with its own set of extracted features. Thus, Table 6.1 includes performance values with decimal points for [24] and [25], while we provide a value for each circuit (i.e., RS232).

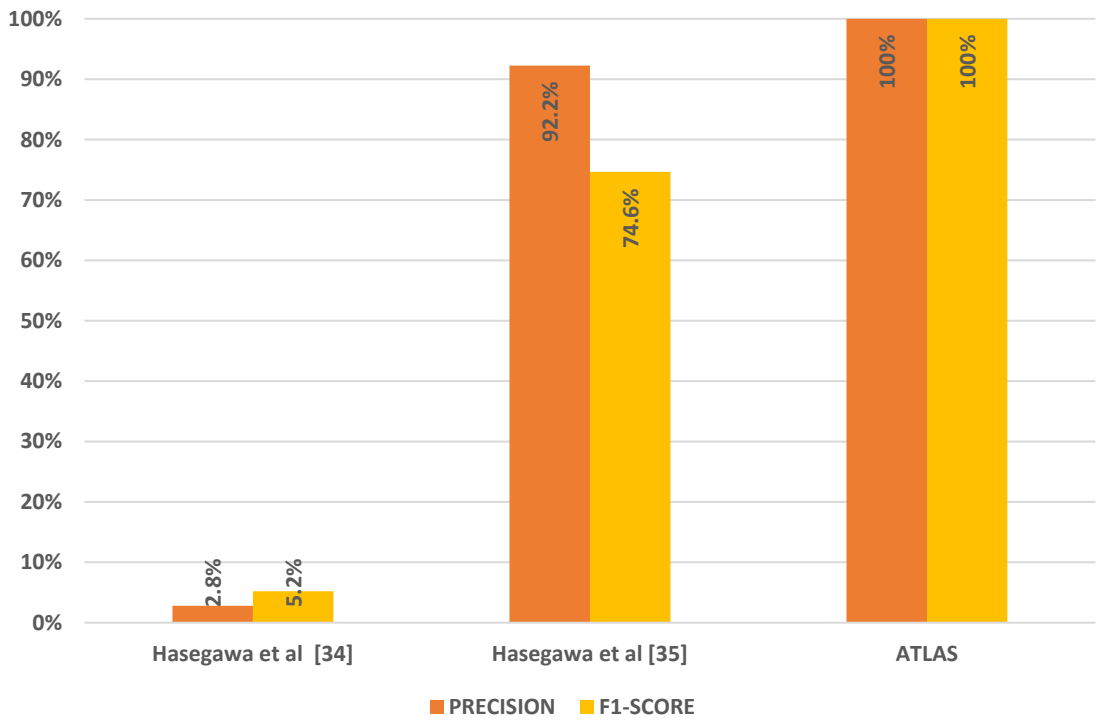


Figure 6.8 Histograms with the performance comparison between existing approaches and our approach ATLAS.

Table 6.1 Table with the comparison of our method with existing methods for the same benchmark

Test Circuits	Precision			F1-Score		
	[24]	[25]	**ATLAS**	[24]	[25]	**ATLAS**
RS232-T1000	11.5%	92.3%	100%	19%	96%	100%
RS232-T1100	3.1%	78.3%	100%	5.9%	61%	100%
RS232-T1200	3.4%	100%	100%	6.5%	93.8%	100%
RS232-T1300	3.5%	100%	100%	6.7%	100%	100%
RS232-T1400	4.1%	100%	100%	7.8%	98.9%	100%
RS232-T1500	4.1%	97.4%	100%	7.9%	96.1%	100%
RS232-T1600	3.5%	90%	100%	6.7%	91.5%	100%
s15850-T100	2.9%	95.5%	100%	5.7%	85.7%	100%
s35932-T100	0.5%	100%	100%	1.1%	84.6%	100%
s35932-T200	0.6%	100%	100%	1.2%	15.4%	100%
s35932-T300	0.4%	96.8%	100%	0.7%	88.2%	100%
s38417-T100	0.8%	100%	100%	1.7%	50%	100%
s38417-T200	0.8%	100%	100%	1.5%	63.6%	100%
s38417-T300	2.6%	100%	100%	5.1%	85.7%	100%
s38584-T100	0.3%	33.3%	100%	0.6%	9.1%	100%
Mean	2.81%	92.2%	100%	5.21%	74.6%	100%

Chapter 7 Conclusions and Future Work

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 data sets.

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 [158][159][160] Our GAINESIS tool is available through this link: https://caslab.e-ce.uth.gr/ToolsandDatabases.html.

References

[1] F. Plessas and G. Kalivas, "A subharmonically injected phase-locked loop for 5-GHz applications", *Microwave and Optical Technology Letters*, 2006, vol. 48, pp. 2158-2162, doi: 10.1002/mop.21888.

[2] F. Plessas, A. Papalambrou, and G. Kalivas, "Subharmonic injection-locking and self-oscillating mixer", *In Proceedings of the 2007 IEEE International Symposium on Circuits and Systems*, New Orleans, LA, USA, 27-30 May 2007, doi: 10.1109/ISCAS.2007.377952.

[3] E. Lourandakis, F. Plessas, and G. Kalivas, "A 0.5 - 5.5 GHz Distributed Low Noise Amplifier", *ECTI Transactions on Electrical Engineering and Electronics and Communication*, 2008, vol. 6, no. 1, pp. 26–31

[4] F. C. Plessas, A. Papalambrou, and G. Kalivas, "A 5-GHz subharmonic injection-locked oscillator and self-oscillating mixer", *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2008, vol. 55, pp. 633-637, doi: 10.1109/TCSII.2008.921575.

[5] A. Tsitouras and F. Plessas, "Ultra wideband, low-power, 3-5.6 GHz, CMOS voltage-controlled oscillator", *Microelectronics Journal,* 2009, vol. 40, pp. 897-904, doi: 10.1016/j.mejo.2009.01.009.

[6] A. Tsitouras and F. Plessas, "Ultra-wideband, low-power, inductorless, 3.1-4.8 GHz, CMOS VCO", *Circuits, Systems, and Signal Processing*, 2011, vol. 30, no. 2, pp. 263-285, doi: 10.1007/s00034-010-9220-6.

[7] F. Plessas, "A study of superharmonic injection locking in multiband frequency dividers", *International Journal of Circuit Theory and Applications*, 2011, vol. 39, pp. 397-410, doi: 10.1002/cta.644.

[8] F. Plessas, A. Tsitouras, and G. Kalivas, "Phase noise characterization of subharmonic injection locked oscillators", *International Journal of Circuit Theory and Applications*, 2011, vol. 39, pp 791-800, doi: 10.1002/cta.734.

[9] A. Tsitouras, F. Plessas, and G. Kalivas, "A linear, ultra wideband, low-power, 2.1-5 GHz, VCO", *International Journal of Circuit Theory and Applications*, 2011, vol. 39,

pp. 823-833, doi: 10.1002/cta.670.

[10] F. Plessas, A. Tsitouras, and G. Kalivas, "5-GHz fully differential multifunctional circuit", *International Journal of Electronics,* 2012, vol. 99, pp. 1317-1322, doi: 10.1080/00207217.2012.669711.

[11] A. Alexandropoulos, E. Davrazos, F. Plessas, and M. Birbas, "A novel 1.8 V, 1066 Mbps, DDR2, DFI-compatible, memory interface", *In Proceedings of the 2010 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Lixouri, Greece, 5-7 July 2010, doi: 10.1109/ISVLSI.2010.49.

[12] A. Alexandropoulos, F. Plessas, M. Birbas, and S. A. Analogies, "A dynamic DFI-compatible strobe qualification system for double data rate (DDR) physical interfaces", *In Proceedings of the 2010 17th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Athens, Greece, 12-15 December 2010, doi: 10.1109/ICECS.2010.5724507.

[13] G. Giannakas, F. Plessas, G. Nassopoulos, and G. Stamoulis, "A 2.45GHz power harvesting circuit in 90nm CMOS", *In Proceedings of the 2010 17th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Athens, Greece, 12-15 December 2010, doi: 10.1109/ICECS.2010.5724642.

[14] F. Plessas and N. Terzopoulos, "60 GHz Millimeter-Wave WLANs and WPANs: Introduction, system design, and PHY layer challenges", *System-Level Design Methodologies for Telecommunication*, 2014, pp. 63-78.

[15] N. Terzopoulos, C. Laoudias, F. Plessas, G. Souliotis, S. Koutsomitsos, and M. Birbas, "A 5-Gbps USB3.0 transmitter and receiver linear equalizer", *International Journal of Circuit Theory and Applications*, 2015, vol. 43, pp. 900–916, doi: 10.1002/cta.1982.

[16] S. Bhunia et al., "Protection against hardware trojan attacks: Towards a comprehensive solution", *IEEE Design & Test*, 2013, vol. 30, pp. 6–17, doi: 10.1109/MDT.2012.2196252.

[17] S. Mitra, H. S. P. Wong, and S. Wong, "The Trojan-proof chip", *IEEE Spectrum*, 2015, vol. 52, pp. 46-51, doi: 10.1109/MSPEC.2015.7024511.

[18] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of Research and Development*, 2000, vol. 3, pp. 210-229, doi:

10.1147/rd.441.0206.

[19] Y. LeCun, Y. Bengio, G. Hinton , “Deep learning”, *Nature*, 2015, vol. 521, pp. 436-444, doi:10.1038/nature14539.

[20] G. K. Georgakilas, A. Grioni, K. G. Liakos, E. Chalupova, F. C. Plessas, and P. Alexiou, “Multi-branch Convolutional Neural Network for Identification of Small Non-coding RNA genomic loci”, *Scientific Reports*, 2020, vol. 10, pp. 9486, doi: 10.1038/s41598-020-66454-3.

[21] X. E. Pantazi, D. Moshou, and A. A. Tamouridou, “Automated leaf disease detection in different crop species through image features analysis and One Class Classifiers”, *Computers and Electronics in Agriculture*, 2019, vol. 156, pp. 96-104, doi: 10.1016/j.compag.2018.11.005.

[22] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, N. Sklavos, and F. C. Plessas, “Conventional and machine learning approaches as countermeasures against hardware trojan attacks”, *Microprocessors and Microsystems*, 2020, vol. 79, pp. 103295, doi: 10.1016/j.micpro.2020.103295.

[23] K. G. Liakos, G. K. Georgakilas, S. Moustakidis, P. Karlsson, and F. C. Plessas, “Machine Learning for Hardware Trojan Detection: A Review”, *In Proceedings of the 2019 Panhellenic Conference on Electronics & Telecommunications (PACET)*, Volos, Greece, 8-9 November 2019, doi: 10.1109/PACET48583.2019.8956251.

[24] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, “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, doi: 10.1109/IOLTS.2016.7604700.

[25] K. Hasegawa, M. Yanagisawa, and N. Togawa, “Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier”, *In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD, USA, 28-31 May 2017, doi: 10.1109/ISCAS.2017.8050827.

[26] T. Inoue, K. Hasegawa, M. Yanagisawa, and N. Togawa, “Designing hardware trojans and their detection based on a SVM-based approach”, *In Proceedings of the 2017*

IEEE 12nd International Conference on ASIC (ASICON), Guiyang, China, 25-28 October 2017, doi: 10.1109/ASICON.2017.8252600.

[27] K. G. Liakos, G. K. Georgakilas, and F. C. Plessas, "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, doi: 10.1109/ISVLSI51109.2021.00081.

[28] H. Salmani, M. Tehranipoor, and R. Karri, "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, doi: 10.1109/ICCD.2013.6657085.

[29] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of Hardware Trojans and Maliciously Affected Circuits", *Journal of Hardware and Systems Security,* 2017, pp. 85-102, doi: 10.1007/s41635-017-0001-6.

[30] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation", *arxiv*, 2018, arxiv:1710.10196.

[31] H. Zhang et al., "StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks", *In Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, 22-29 October 2017, doi: 10.1109/ICCV.2017.629.

[32] Y. Li, S. Liu, J. Yang, and M. H. Yang, "Generative face completion", *In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5892-5900, doi: 10.1109/CVPR.2017.624.

[33] H. Zhang, V. Sindagi, and V. M. Patel, "Image De-Raining Using a Conditional Generative Adversarial Network", *IEEE Transactions on Circuits and Systems for Video Technology, 2020*, vol. 30, pp. 3943-3956, doi: 10.1109/TCSVT.2019.2920407.

[34] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures", *Proceedings of the IEEE*, 2014, vol. 102, pp. 1229-1247, doi: 10.1109/JPROC.2014.2334493.

[35] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware

automatically", *In Proceedings of the 2010 IEEE Symposium on Security and Privacy (SSP)*, Oakland, CA, USA, 16-19 May 2010, doi: 10.1109/SP.2010.18.

[36] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou, "Designing and implementing malicious hardware", *In Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (UWLSEET)*, San Francisco, CA, USA 15 April 2008.

[37] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection", *IEEE Design and Test of Computers*, 2010, vol. 27, pp. 10-25, doi: 10.1109/MDT.2010.7.

[38] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans", *Computer*, 2010, vol. 43, pp. 39-46, doi: 10.1109/MC.2010.299.

[39] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, 1943, vol. 5, pp. 115-133,doi: 10.1007/BF02478259.

[40] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review*, 1958, vol. 65, pp. 386-408, doi: 10.1037/h0042519.

[41] S. K. Pal and S. Mitra, "Multilayer Perceptron, Fuzzy Sets, and Classification", *IEEE Transactions Neural Networks*, 1992, vol. 3, pp. 683-697, doi: 10.1109/72.159058.

[42] H. J. Kelley, "Gradient Theory of Optimal Flight Paths", *American Rocket Society Journal*, 1960, vol. 30, doi: 10.2514/8.5282.

[43] M. Riedmiller and H. Braun, "Direct adaptive method for faster backpropagation learning: The RPROP algorithm", *In Proceedings of the IEEE International Confernce on neural Networks (ICNN)*, San Francisco, CA, USA, 28 March-1 April 1993, doi: 10.1109/icnn.1993.298623.

[44] R. Hecht-Nielsen, "Applications of counterpropagation networks", *Neural Networks*, 1988, vol. 1. pp. 131-139, doi: 10.1016/0893-6080(88)90015-9.

[45] D. Broomhead, D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems*, 1988, vol. 2, pp. 321–355.

[46] W. Melssen, R. Wehrens, and L. Buydens, “Supervised Kohonen networks for classification problems”, *Chemometrics and Intelligent Laboratory Systems,* 2006, vol. 83, pp. 99–113, doi: 10.1016/j.chemolab.2006.02.003.

[47] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities.”, *Proceedings of the National Academy Sciences of the U. S. A.*, 1982, vol. 79, pp. 2554-2558, doi: 10.1073/pnas.79.8.2554.

[48] D. F. Specht, “A General Regression Neural Network”, *IEEE Transactions on Neural Networks*, 1991, vol. 2, pp. 568-576, doi: 10.1109/72.97934.

[49] C. Y. Liou, W. C. Cheng, J. W. Liou, and D. R. Liou, “Autoencoder for words”, *Neurocomputing*, 2014, vol. 139, pp. 84-96, doi: 10.1016/j.neucom.2013.09.055.

[50] J. S. R. Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference System”, *IEEE Transactions on Systems, Man, and Cybernetics,* 1993, vol. 23, pp. 665-685, doi: 10.1109/21.256541.

[51] G. Bin Huang, Q. Y. Zhu, and C. K. Siew, “Extreme learning machine: Theory and applications”, *Neurocomputing*, 2006, vol. 70, pp. 489-501, doi: 10.1016/j.neucom.2005.12.126.

[52] J. Cao, Z. Lin, and G. Bin Huang, “Self-adaptive evolutionary extreme learning machine”, *Neural Processing Letters*, 2012, vol. 36, pp. 285-305, doi: 10.1007/s11063-012-9236-y.

[53] J. Pearl, “Probabilistic reasoning in intelligent systems: Networks of plausible inference”, *Morgan Kaufmann Publishers*, 1988, doi: 10.5555/534975.

[54] R. E. Neapolitan, “Models for reasoning under uncertainty”, *Applied Artificial Intelligence*, 2007, vol. 1, pp. 337-336, doi: 10.1080/08839518708927979.

[55] A. Ligeza, “Artificial Intelligence: A Modern Approach”, *Neurocomputing*, 1995, vol. 9, pp. 215–218, doi: 10.1016/0925-2312(95)90020-9.

[56] K. Ali, A. Jamali, M. Abbas, K. Ali Memon, and A. Aleem Jamali, “Multinomial Naive Bayes Classification Model for Sentiment Analysis”, *IJCSNS International Journal of Computer Science and Network Security*, 2019, vol. 19, pp. 62-67.

[57] M. Ontivero-Ortega, A. Lage-Castellanos, G. Valente, R. Goebel, and M. Valdes-Sosa, “Fast Gaussian Naïve Bayes for searchlight classification analysis”,

Neuroimage, 2017, vol. 163, pp. 471-479, doi: 10.1016/j.neuroimage.2017.09.001.

[58] R. C. Tryon, "Communality of a variable: Formulation by cluster analysis", *Psychometrika*, 1957, vol. 22, pp. 241-260, doi: 10.1007/BF02289125.

[59] S. P. Lloyd, "Least Squares Quantization in PCM", *IEEE Transactions on Information Theory*, 1982, vol. 28, pp. 129-137, doi: 10.1109/TIT.1982.1056489.

[60] S. C. Johnson, "Hierarchical clustering schemes", *Psychometrika*, 1967, vol. 32, pp. 241-254, doi: 10.1007/BF02289588.

[61] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the EM Algorithm", *Journal of the Royal Statistical Society. Series B*, 1977, vol. 39, pp. 1–22, doi: 10.1111/j.2517-6161.1977.tb01600.x.

[62] Y. Yuan, X. Chen, X. Chen, and J. Wang, "Segmentation Transformer: Object-Contextual Representations for Semantic Segmentation", *Computer Vision – ECCV*, 2020, pp. 173-190, doi:10.1007/978-3-030-58539-6_11.

[63] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, "Fixing the train-test resolution discrepancy", *Advances in Neural Information Processing Systems 32* (NeurIPS 2019), 2019.

[64] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection", *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10781-10790, doi: 10.1109/CVPR42600.2020.01079.

[65] W. A. Belson, "Matching and Prediction on the Principle of Biological Classification", *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1959, vol. 8, pp. 65-75, doi: 10.2307/2985543.

[66] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees", 2017, pp. 368, doi: 10.1201/9781315139470.

[67] G. V. Kass, "An Exploratory Technique for Investigating Large Quantities of Categorical Data", *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1980, vol. 29, pp. 119-127, doi: 10.2307/2986296.

[68] A. M. Hormann, "Programs for machine learning Part I", *Information and Control*, 1962, vol. 5, pp. 2347-367, doi: 10.1016/S0019-9958(62)90649-6.

[69] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, 2015, vol 521, pp.

436-444, doi: 10.1038/nature14539.

[70] N. Milosevic, “Introduction to Convolutional Neural Networks”, 2020, doi: 10.1007/978-1-4842-5648-0.

[71] R. Salakhutdinov and G. Hinton, “Deep Boltzmann machines”, *In Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics (ICAIS)*, 2009, pp. 448-455.

[72] Y. Hua, J. Guo, and H. Zhao, “Deep Belief Networks and deep learning”, *In Proceedings of the 2015 International Conference on Intelligent Computing and Internet of Things (ICICIT),* 2015, doi: 10.1109/ICAIOT.2015.7111524.

[73] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, “Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”, *The Journal of Machine Learning Research*, 2010, vol. 11, pp. 3371-3408.

[74] L. R. Medsker and L. C. Jain, “Recurrent Neural Networks Design and Applications”, *Journal of Chemical Information and Modeling*, 2013.

[75] S. Hochreiter and J. Schmidhuber, “Long Short Term Memory. Neural Computation”, *Neural Computation*, 1997, vol. 9, pp. 1735-1780, doi: 10.1162/neco.1997.9.8.1735.

[76] K. Pearson, “LIII. On lines and planes of closest fit to systems of points in space”, *London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* 1901, vol. 2, pp. 559-572, doi: 10.1080/14786440109462720.

[77] A. Leguina, “A primer on partial least squares structural equation modeling (PLS-SEM)”, *International Journal of Research & Method in Education*, 2015, vol. 38, pp. 220-221, doi: 10.1080/1743727x.2015.1005806.

[78] P. Sarkar, “What is LDA: Linear Discriminant Analysis for Machine Learning”, *Knowledge Hut*, 2019.

[79] R. E. Schapire, “Explaining adaboost”, *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, 2013, pp. 37-52, doi: 10.1007/978-3-642-41136-6_5.

[80] L. Breiman, “Bagging predictors”, *Machine Learning*, 1996, vol. 24, pp 123-140, doi: 10.1007/bf00058655.

[81] R. E. Schapire, “A brief introduction to boosting”, *In Proceedings of the 16th*

International Joint Conference on Artificiasl Intelligent (IJCAI), 1999.

[82] J. H. Friedman, "Greedy function approximation: A gradient boosting machine", *The Annals of Statistics*, 2001, vol. 29, pp. 1189-1232, doi: 10.1214/aos/1013203451.

[83] L. Breiman, "Random forests", *Machine Learning*, 2001, vol. 45, pp. 5-32, doi: 10.1023/A:1010933404324.

[84] I. Goodfellow et al., "Generative adversarial networks", *Communications of the ACM*, 2020, vol. 63, pp. 139-144, doi: 10.1145/3422622.

[85] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets", *arxiv*, 2014, vol. 1, pp. 1-7, arxiv: 1411.1784.

[86] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN", *arxiv*, 2017, vol.1, pp 1-32, arxiv:1701.07875.

[87] S. Qin and T. Jiang, "Improved Wasserstein Conditional Generative Adversarial Network Speech Enhancement", *EURASIP Journal on Wireless Communications and Networking*, 2018, doi: 10.1186/s13638-018-1196-0.

[88] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks", *In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, 18-20 June 2019, doi: 10.1109/CVPR.2019.00453.

[89] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks", *In Proceedings of the International Conference on Computer Vision (ICCV)*, Venice, Italy, 22-29 October 2017, pp. 2223-2232, doi: 10.1109/ICCV.2017.244.

[90] E. Fix and J. L. Hodges, "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties", *International Statistical Review / Revue Internationale de Statistique*, 1989, vol. 57, p. 238, doi: 10.2307/1403797.

[91] T. Kohonen, "Statistical Pattern Recognition Revisited", *Advanced Neural Computers*, 1990, pp. 137-144, doi: 10.1016/B978-0-444-88400-8.50020-0.

[92] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning", *Artificial Intelligence Review*, 1997, vol. 11, pp. 11-73, doi: 10.1007/978-94-017-2053-3_2.

[93] C. Cortes and V. Vapnik, "Support-Vector Networks", *Machine Learning*, 1995, vol.

20, pp. 273-297, doi: 10.1023/A:1022627411411.

[94] T. Kohonen, "The self-organizing map," *Neurocomputing*, 1998, vol. 21, pp. 1–6, doi: 10.1016/S0925-2312(98)00030-7.

[95] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Naacl-Hlt)*, Minneapolis, MI, USA, June 2019, doi: 10.18653/v1/N19-1423.

[96] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding", *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019, doi: 10.48550/arXiv.1906.08237.

[97] K. Park, R. Rothfeder, S. Petheram, F. Buaku, R. Ewing, and W. H. Greene, "Linear regression", *Basic Quantitative Research Methods for Urban Planners*, 2020.

[98] D. R. Cox, "The Regression Analysis of Binary Sequences", *Journal of the Royal Statistical Society. Series B,* 1959, vol. 21, pp. 238–238, doi: 10.1111/j.2517-6161.1959.tb00334.x.

[99] G. Hutcheson, "Ordinary Least-Squares Regression", *The SAGE Dictionary of Quantitative Management Research,* 2011, pp. 224-228.

[100] J. R. Quinlan, "Learning with continuous classes", *In Proceedings of the Australian Joint Conference on Artificial Intelligence,* Hobart, Australia, 16-18 November 1992, pp. 343-348.

[101] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots", *Journal of the American Statistical Association,* 1979, vol. 74, pp. 829-836, doi: 10.1080/01621459.1979.10481038.

[102] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems", *Technometrics*, 1970, vol. 42, pp. 80-86, doi: 10.1080/00401706.1970.10488634.

[103] R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso", *Journal of the Royal Statistical Society. Series B,* 1996, vol. 58, pp. 267-288, doi: 10.1111/j.2517-6161.1996.tb02080.x.

[104] B. Efron et al., “Least angle regression”, *Annuals of Statistics*, 2004, vol. 32, pp. 407-499, doi: 10.1214/009053604000000067.

[105] W. Han et al., “ContextNet: Improving convolutional neural networks for automatic speech recognition with global context”, *In Proceedings of the Conference of the International Speech communication Association (INTERSPEECH),* Shanghai, China, 25-29 October 2020, doi: 10.21437/Interspeech.2020-2059.

[106] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, “Light Gated Recurrent Units for Speech Recognition”, *IEEE Transactions on Emerging Topics in Computational Intelligence,* 2018, vol. 1, doi: 10.1109/TETCI.2017.2762739.

[107] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition”, *In Proceedings of the IEEE Conference on Computer Vision and Pattern (CVPR)*, Las Vegas, NE, USA, 26 June - 1 July 2016, doi: 10.1109/CVPR.2016.90.

[108] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, “Trojan detection using IC fingerprinting”, *In Proceedings of the IEEE Symposium on Security and Privacy (SP '07),* Berkeley, CA, USA, 20-23 May 2007, doi: 10.1109/SP.2007.36.

[109] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, “MERO: A statistical approach for hardware Trojan detection”, *Lecture Notes in Compuer Science*, 2009, vol. 5747 LNCS, pp. 396–410, doi: 10.1007/978-3-642-04138-9_28.

[110] H. Salmani, M. Tehranipoor, and J. Plusquellic, “A Novel Technique for Improving Hardware Trojan Detection and Reducing Trojan Activation Time”, *IEEE Transactions on Very Large Scale Integration Systems,* vol. 20, pp. 112–125, Jan. 2012, doi: 10.1109/TVLSI.2010.2093547.

[111] C. Bao, D. Forte, and A. Srivastava, “On application of one-class SVM to reverse engineering-based hardware Trojan detection”, *In Proceedings of the 15th International Symposium on Quality Electronic Design,* Santa Clara, CA, USA, 3-5 March 2014, doi: 10.1109/ISQED.2014.6783305.

[112] X. T. Ngo, J. L. Danger, S. Guilley, Z. Najm, and O. Emery, “Hardware property checker for run-time Hardware Trojan detection”, *In Proceedings of the 2015 European Conference on Circuit Theory and Design (ECCTD),* Trondheim, Norway, 24-26 August 2015, doi: 10.1109/ECCTD.2015.7300085.

[113] K. G. Liakos, G. K. Georgakilas, F. C. Plessas, and P. Kitsos, “GAINESIS:

Generative Artificial Intelligence NEtlists SynthesIS", *Electronics*, 2022, vol. 11, pp. 245, doi: 10.3390/electronics11020245.

[114] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic, "Detecting trojans through leakage current analysis using multiple supply pad IDDQs", *IEEE Transactions on. Information Forensics Security,* 2010, vol. 5, pp. 893-904, doi: 10.1109/TIFS.2010.2061228.

[115] R. Rad, J. Plusquellic, and M. Tehranipoor, "A sensitivity analysis of power signal methods for detecting hardware trojans under real process and environmental conditions", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* 2010, vol. 18, pp. 1735-1744, doi: 10.1109/TVLSI.2009.2029117.

[116] F. Koushanfar and A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits trojan detection", *IEEE Transactions on Information Forensics and Security*, 2011, vol. 6, pp. 162-174, doi: 10.1109/TIFS.2010.2096811.

[117] S. Narasimhan et al., "Hardware trojan detection by multiple-parameter side-channel analysis", *IEEE Transactions on Computers,* 2013, vol. 62, pp. 2183-2195, doi: 10.1109/TC.2012.200.

[118] C. Lamech, R. M. Rad, M. Tehranipoor, and J. Plusquellic, "An experimental analysis of power and delay signal-to-noise requirements for detecting trojans and methods for achieving the required detection sensitivities", *IEEE Transactions on Information Forensics and Security*, 2011, vol. 6, pp. 1170-1179, doi: 10.1109/TIFS.2011.2136339.

[119] K. Xiao, X. Zhang, and M. Tehranipoor, "A clock sweeping technique for detecting hardware trojans impacting circuits delay", *IEEE Design & Test, 2013*, vol. 30, pp. 26–34, doi: 10.1109/MDAT.2013.2249555.

[120] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis", *In Proceedings of the 2013 ACM SIGSAC Confernce on Computer & Communications Security,* Berlin, Germany, 4 November 2013, pp. 697-708, doi: 10.1145/2508859.2516654.

[121] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, vol. 34, pp. 1148-1161, doi: 10.1109/TCAD.2015.2422836.

[122] C. Bao, Y. Xie, Y. Liu, and A. Srivastava, “On Reverse Engineering-Based Hardware Trojan Detection”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 2016, vol. 35, pp. 49-57, doi: 10.1109/TCAD.2015.2488495.

[123] D. Jap, W. He, and S. Bhasin, “Supervised and unsupervised machine learning for side-channel based Trojan detection”, *In Proceedings of the 27th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, London, UK, 06-08 July 2016, doi: 10.1109/ASAP.2016.7760768.

[124] M. Xue, J. Wang, and A. Hux, “An enhanced classification-based golden chips-free hardware Trojan detection technique”, *In Proceedings of the 2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, Yilan, Taiwan, 19-20 December 2016, doi: 10.1109/AsianHOST.2016.7835553.

[125] S. Wang, X. Dong, K. Sun, Q. Cui, D. Li, and C. He, “Hardware Trojan detection based on ELM neural network”, *In Proceedings of the First IEEE International Conference on Computer Communication and the Internet (ICCCI),* Wuhan, China, 13-15 October 2016, doi: 10.1109/CCI.2016.7778952.

[126] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, “Detection technique for hardware Trojans using machine learning in frequency domain”, *In Proceedings of the 4th IEEE Global Conference on Consumer Electronics (GCCE)*, Osaka, Japan, 27-30 October 2015, doi: 10.1109/GCCE.2015.7398569.

[127] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, “Silicon Demonstration of Hardware Trojan Design and Detection in Wireless Cryptographic ICs”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017, vol. 25, pp. 1506-1519, doi: 10.1109/TVLSI.2016.2633348.

[128] F. Khalid, S. R. Hasan, O. Hasan, and F. Awwad, “Runtime hardware Trojan monitors through modeling burst mode communication using formal verification”, *Integration*, 2018, vol. 61, pp. 62–76, doi: 10.1016/j.vlsi.2017.11.003.

[129] C. Bao, D. Forte, and A. Srivastava, “Temperature Tracking: Toward Robust Run-Time Detection of Hardware Trojans”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, vol. 34, no. 10, pp. 1577–1585, doi: 10.1109/TCAD.2015.2424929.

[130] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, “LUT-Lock: A novel

LUT-based logic obfuscation for FPGA-Bitstream and ASIC-hardware protection", *In Proceedings of the 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Hong Kong, China, 09 August 2018, doi: 10.1109/ISVLSI.2018.00080.

[131] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-Lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks", *In Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, USA, 2-6 June 2019, doi: 10.1145/3316781.3317831.

[132] B. Khaleghi, A. Ahari, H. Asadi, and S. Bayat-Sarmadi, "FPGA-based protection scheme against hardware trojan horse insertion using dummy logic", *IEEE Embeded Systems Letters*, 2015, vol. 7, pp. 46–50, doi: 10.1109/LES.2015.2406791.

[133] A. Nejat, S. M. H. Shekarian, and M. Saheb Zamani, "A study on the efficiency of hardware Trojan detection based on path-delay fingerprinting", *Microprocessors and Microsystems*, 2014, vol. 38, pp. 246–252, doi: 10.1016/j.micpro.2014.01.003.

[134] S. M. H. Shekarian and M. Saheb Zamani, "Improving hardware Trojan detection by retiming", *Microprocessors and Microsystems*, 2015, vol. 39, pp. 145-156, doi: 10.1016/j.micpro.2015.02.002.

[135] M. Pilgrim, S. Willison, "Dive Into Python", *Springer*, 2009, vol. 2.

[136] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning", *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, Savannah, USA, 2-4 November 2016, doi: 10.5555/3026877.3026899.

[137] F. Chollet, "Keras", *Journal of Chemical Information and Modeling*, 2015.

[138] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," The *Journal of Machine Learning Research*, 2011, vol. 12, pp. 2825–2830.

[139] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California, USA, 13-17 August 2016, doi: 10.1145/2939672.2939785.

[140] T. Kluyver *et al.*, "Jupyter Notebooks—a publishing format for reproducible computational workflows", *In 20th International Conference on Electronic Publishing (ELPUB),* Göttingen, Germany, 7-9 June 2016, pp. 87-90, doi:

10.3233/978-1-61499-649-1-87.

[141] C. H. M. Oliveira, M. T. Moreira, R. A. Guazzelli, and N. L. V. Calazans, "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, 06 February 2016, doi: 10.1109/ICECS.2016.7841286.

[142] C. W. Royer, M. O'Neill, and S. J. Wright, "A Newton-CG algorithm with complexity guarantees for smooth unconstrained optimization", *Mathematical Programming*, 2020, vol. 180, pp. 451-488, doi: 10.1007/s10107-019-01362-7.

[143] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization", *Mathematical Programming*, 1989, vol.45, pp. 503-528, doi: 10.1007/BF01589116.

[144] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin, "LIBLINEAR: A library for large linear classification", *The Journal of Machine Learning Research*, 2008, vol. 9, pp. 1871-1874, doi: 10.5555/1390681.1442794.

[145] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient", arxiv, 2017, pp. 1-52, arxiv:1309.2388.

[146] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives", *arxiv*, 2014, pp. 1-15, arxiv:1407.0202.

[147] J. A. K. Suykens and J. Vandewalle, "Least Squares Support Vector Machine Classifiers", *Neural Processing Letters*, 1999, vol. 9, pp. 293–300, doi: 10.1023/A:1018628609742.

[148] C. C. Chang and C. J. Lin, "LIBSVM: A Library for support vector machines", *ACM Transactions on Intelligent Systems and Technology*, 2011, vol. 2, pp.1-27, doi: 10.1145/1961189.1961199.

[149] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression", *Statistics and Computing*, 2004, vol. 14, pp. 199-222, doi: 10.1023/B:STCO.0000035301.49549.88.

[150] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, "Basic Methods of Least Squares Support Vector Machines," *Least Squares Support*

Vector Machines, 2002, pp. 71–116, doi.org/10.1142/9789812776655_0003.

[151] R. Galvão, M. Araújo, W. Fragoso, E. Silva, G. José, S. Soares, H. Paiva, "A variable elimination method to improve the parsimony of MLR models using the successive projections algorithm", *Chemometrics and Intelligent Laboratory Systems*, 2008, vol. 92, pp. 83-91, doi: 10.1016/j.chemolab.2007.12.004.

[152] S. Ruder, "An overview of gradient descent optimization algorithms", *arxiv,* 2017, pp. 1-14 , arxiv: 1609.04747.

[153] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization", *arxiv*, 2015, pp. 1-15, arxiv:1412.6980.

[154] T. Kurbiel and S. Khaleghian, "Training of Deep Neural Networks based on Distance Measures using RMSProp", *arxiv*, 2017, pp. 1–6, arxiv:1708.01911.

[155] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)", *arxiv*, 2018, pp. 2–8, arxiv:1803.08375.

[156] J. Han and C. Moraga, "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, doi:10.1007/3-540-59497-3_175.

[157] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning", *arxiv*, 2018, pp. 1–20, arxiv:1811.03378.

[158] P. Kitsos, D. E. Simos, J. Torres-Jimenez, and A. G. Voyiatzis, "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; doi:10.1109/ISSRE.2015.7381800.

[159] L. Pyrgas and P. Kitsos, "A hybrid FPGA trojan detection technique based-on combinatorial testing and on-chip sensing", *In Proceedings of the 2018 Springer 14th International Symposium on Applied Reconfigurable Computing (ARC)*, Santorini, Greece, 2-4 May 2018, doi: 10.1007/978-3-319-78890-6_24.

[160] A. P. Fournaris, L. Pyrgas, and P. Kitsos, "An efficient multi-parameter approach for FPGA hardware Trojan detection", *Microprocess and Microsystems*, 2019, vol 71, doi: 10.1016/j.micpro.2019.102863.