IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 18, 2022, accepted April 28, 2022, date of publication May 9, 2022, date of current version May 12, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3173287

A Survey on Hardware Vulnerability Analysis Using Machine Learning

ZHIXIN PAN, (Member, IEEE), AND PRABHAT MISHRA, (Fellow, IEEE)
Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611, USA
Corresponding author: Zhixin Pan (panzhixin@ufl.edu)

This work was supported in part by the U.S. National Science Foundation (NSF) under Grant CCF-1908131 and Grant SaTC-1936040.

ABSTRACT Electronic systems rely on efficient hardware, popularly known as system-on-chip (SoC), to support its core functionalities. A typical SoC consists of diverse components gathered from third-party vendors to reduce SoC design cost and meet time-to-market constraints. Unfortunately, the participation of third-party companies in global supply chain introduces potential security vulnerabilities. There is a critical need to efficiently detect and mitigate hardware vulnerabilities. Machine learning has been successfully used in hardware security verification as well as development of effective countermeasures. There are recent surveys on hardware Trojan detection using machine learning. To the best of our knowledge, there are no comprehensive surveys on utilization of machine learning techniques for detection and mitigation of a wide variety of hardware vulnerabilities including malicious implants (e.g., hardware Trojans), side-channel leakage, reverse engineering, and supply-chain vulnerabilities (e.g., counterfeiting, overbuilding and recycling). In this paper, we provide a comprehensive survey of hardware vulnerability analysis using machine learning techniques. Specifically, we discuss how existing approaches effectively utilize machine learning algorithms for hardware security verification using simulation-based validation, formal verification as well as side-channel analysis.

INDEX TERMS Hardware security, machine learning, embedded system, vulnerability analysis.

I. INTRODUCTION

System-on-Chip (SoC) is the brain behind a vast majority of electronic devices today. Even resource constrained Internet-of-Things (IoT) devices nowadays incorporate one or more complex SoCs. SoC incorporates a wide variety of components in a single *integrated circuit* (IC) A typical SoC design consists of multiple Intellectual Property (IP) cores including processor, memory, network-on-chip, controllers, converters, input/output devices, etc.

Drastic increase in SoC complexity has led to significant increase in SoC design and validation complexity. Semiconductor companies utilize global supply chain during SoC design and manufacturing to reduce cost and meet time-to-market constraints. Figure 1 from [1] shows a typical supply chain that involves multiple third-party companies. Unfortunately, reliance on third-party IPs raises hardware security concerns. For example, a hardware IP gathered from a potentially untrusted vendor may come with malicious implants (e.g., hardware Trojans), backdoor for information leakage, or other integrity issues. Since application software relies on the hardware root-of-trust for providing any security guarantees, it is critical to ensure that the underlying hardware does not have any security vulnerabilities. In other words, hardware vulnerabilities affect the security and trustworthiness of SoC computing platforms. These vulnerabilities should be fixed before deployment since it affects the overall system security. Based on CVE-MITRE estimates, we can improve the overall system security by 43% by removing hardware-level security vulnerabilities [2].

While there are a wide variety of researches for hardware security verification [3], [4], machine learning has emerged as the feasible solution for efficient detection and mitigation of hardware security vulnerabilities. Figure 2 shows the increasing number of publications in applying machine learning to solve hardware security challenges. This paper provides a comprehensive survey of hardware vulnerability analysis using machine learning. We outline potential security threats and effective machine learning based solutions, while existing surveys related to hardware vulnerability analysis have focused on hardware Trojan detection [5]. Specifically, the topic of hardware Trojan detection, outlined

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

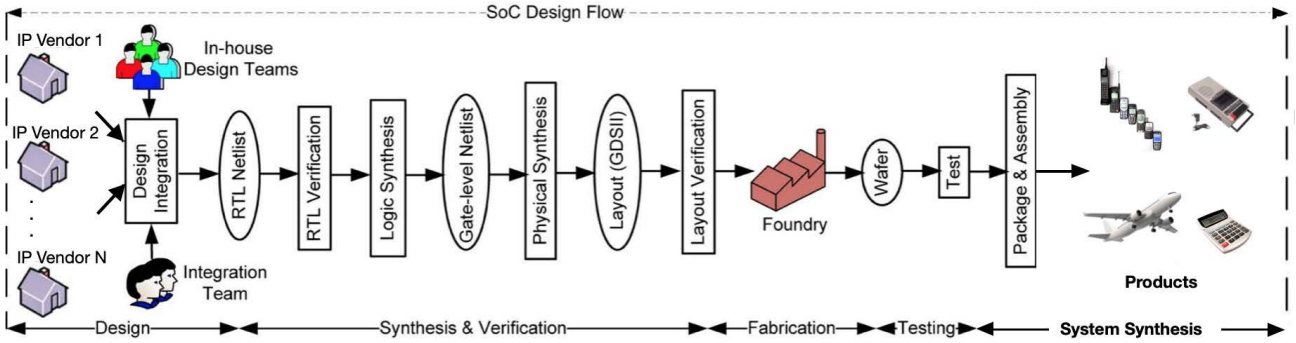

FIGURE 1. Hardware design flow and supply chain distribution. Image credit: [1].

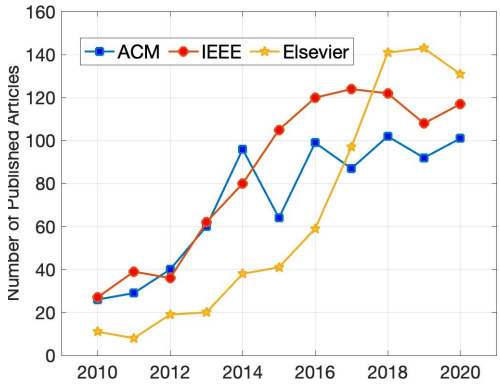

FIGURE 2. Publication trends on ML techniques applied for SoC security from IEEE, ACM, and Elsevier.

in Section III, is a small fraction of overall hardware security landscape. To the best of our knowledge, there is no comprehensive survey of hardware vulnerability analysis using machine learning techniques.

This paper is organized as follows. Section II provides a brief introduction to machine learning. Section III provides an overview of a wide variety of SoC vulnerabilities. Section IV introduces the methodology of this survey. Specifically, it identifies four widely studied categories of SoC vulnerability analysis using machine learning: i) simulation-based validation, ii) formal verification, iii) static analysis of design features, and iv) side-channel analysis. Sections V-VIII describe these four classes of vulnerability analysis. Finally, Section X summarizes the survey and outlines future research directions.

II. OVERVIEW OF MACHINE LEARNING MODELS

With the growing difficulty and resource constraints, recent SoC security study turns to seek assistance from *Machine Learning* (ML) techniques. ML is a class of algorithms that primarily focuses on generating the 'models', namely, learning algorithms, from a large amount of historical 'training data' and then utilizing these trained models for prediction or classification. ML algorithms have enabled promising performance with outstanding flexibility and generalization across various application domains. A typical workflow of ML applied for hardware vulnerability analysis

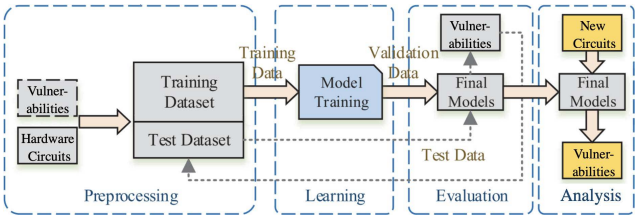

FIGURE 3. The general flow of ML techniques that consists of four major steps: preprocessing, learning, evaluation, and analysis [5].

is depicted in Fig. 3 from [5] that consists of four major steps. The first steps gathers dataset from known hardware vulnerabilities.

This dataset is used to train the ML model in the second step. The trained model is evaluated in the third step. The validated model can be used to detect unknown vulnerabilities in the final step. This section provides an overview of various ML techniques. The subsequent sections will use these ML models for analyzing hardware security vulnerabilities.

A. SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a typical supervised learning model. Intuitively, supervised learning represents a learning process where a ML model is trained to satisfy given training samples with determined labels in advance. After the stage of training using the labeled set, the obtained ML model is expected to respond to new occurrences. For SVM, it is trained to obtain a hyper-plane in data space to separate them out with given labels, while trying to maximize the margin distance between data points and the hyper-plane. A trivial SVM example that achieves a bi-classification between red and green circles is shown in Figure 4. The cyan block in the picture represents the maximized margin for an optimal classifier. Intuitively, the margin distance reflects how 'far' data points from two classes are from each other, and a large margin distance indicates a better chance to correctly classify new occurrences.

B. MULTI-LAYER PERCEPTRON

A commonly used Multilayer Perceptron (MLP) is a feed-forward model. An MLP contains multiple layers of neurons

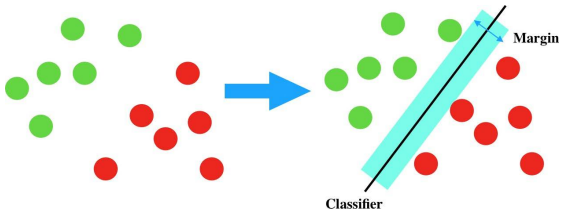

FIGURE 4. Trivial example of SVM.

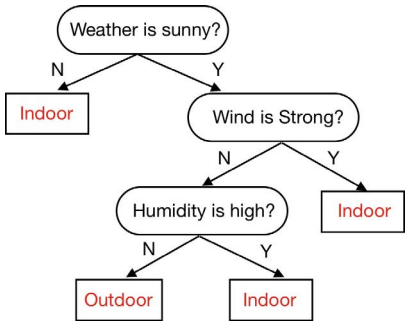

FIGURE 5. An example decision tree model.

with an activation function, each layer fully connects the next layer with numerical values called weights. This activation function maps weighted inputs to the output of the neuron.

The objective of the MLP is to learn these weights for matching the inputs to the outputs as efficiently as possible.

C. DECISION TREE (DT)

Decision Tree (DT) is another commonly used supervised learning algorithm for classification method. An example decision tree is shown in Figure 5 to illustrate its functionality and interpretability. The task is to decide the location for exercise based on weather conditions. As we can see, DT is a tree-structure model with nodes and edges. The basic workflow of DT is to perform a top-down tree traversal from the root. At each node, a specific attribute becomes the key factor to determine the branch. This process continues until a category label is reached at the leaf node, which becomes the final decision. The construction of such a decision tree relies on the selection of attributes for given task. In many real-world applications, this challenge is addressed using a large amount of historical data to automatically build the decision tree. The construction of the tree is usually recursive and can be performed automatically.

D. RANDOM FOREST (RF)

Random Forest (RF), as the name suggested, is an ensemble of decision trees. Strictly speaking, RF is actually an integrated algorithm. It first randomly selects different features and training samples to generate a large number of decision trees, where each decision tree is trained with a subset of the training data. Then RF synthesizes the results of these decision trees by voting or taking average in the ensemble to present the final output. Random forest is widely used in reality analysis. Compared with decision trees, it has

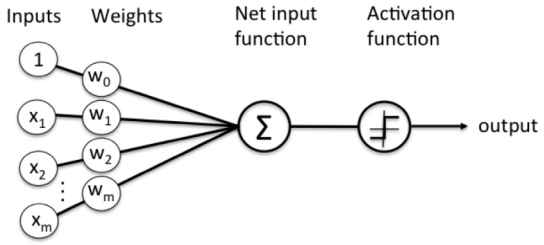

FIGURE 6. A typical DNN neuron. The activation layer is applied to induce non-linearity.

a great improvement in accuracy and robustness at the same time.

E. LINEAR REGRESSION (LR)

The motivation for Linear Regression (LR) comes from statistics. Given data set, LR learns from this data set to produce a linear model that reflects the relationship between x_i and y_i as accurately as possible. Formally, the model can be written as

$$f(\mathbf{x}) = \mathbf{W}^\top \mathbf{x} + b$$

where, $\mathbf{x} = x_1, x_2, \ldots, b$ is the bias term, and $\mathbf{W}$ are weight parameters indicating the weight of corresponding attributes. The learning process of LR is an optimization problem to obtain optimal weight parameters $\mathbf{W}$. The trained model with suitable weights is used to predict values for new inputs.

F. DEEP NEURAL NETWORK (DNN)

Deep Neural Network (DNN) is an artificial neural network which can express or simulate a wide variety of intrinsic functionalities in the fields of classification, regression, re-construction, etc. Analogous to neurons in our nervous system, the 'neural network' of artificial intelligence is a system built from 'neurons' as shown in Figure 6.

The functionality of one single neuron is limited but DNN makes use of multiple neurons and arrange them in layers. Typically, DNN consists of an input layer, output layer and arbitrary numbers of hidden layers in between for enabling DNNs to approximate the complex mapping of given data's inputs and outputs. The training process of DNNs can be summarized as follows

1) Determine the structure and initialize the weights.
2) Feed training samples to compute training loss.
3) Compute the gradient of loss, use backpropagation [6] to update the weight coefficients, and repeat until convergence.

The flexibility of DNNs also enables their different variations to be successfully applied into various context. In this section, we discuss three specific examples of such variations.

1) CONVOLUTION NEURAL NETWORK (CNN)

CNN is a variant of DNN utilizing convolution layers. The emergence of convolutional neural networks has surpassed ordinary neural networks in the field of image processing with

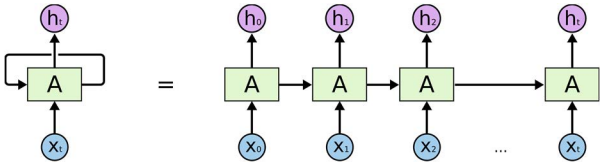

FIGURE 7. Basic structure of RNN.

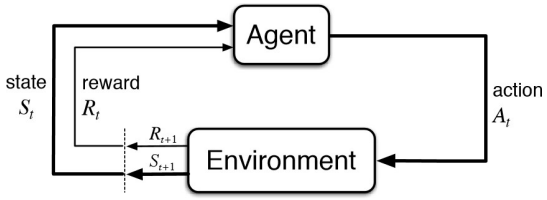

FIGURE 8. The basic framework of reinforcement learning.

the characteristics of fewer parameters, fast training, high scores, and easy migration.

2) RECURRENT NEURAL NETWORK (RNN)

RNN is another variant of neural networks. The general structure of RNN is shown in Figure 7. In the picture, **A** represents the neural network architecture, where x_0, x_1, $x_2, \ldots, x_t$ represents the time series inputs and h_is are the outputs of hidden layers.

For each single input x_i, RNN not only provides immediate response h_i, but also stores the information of the current input by updating the architecture itself. Meanwhile, stored information will also be fed into the architecture in the next iteration to supply extra information. Therefore, it is widely applied in security domain as it is efficient to capture temporal dependencies of signals.

In practical works, a specific type of RNN model called *Long Short-term Memory* (LSTM) is widely adopted. It applies *gate* mechanism to solve vanishing gradient and exploding gradient. Meanwhile, the gate mechanism provides feature filtering, saving useful features and discarding useless features, which greatly enriches the information representation capacity of the model. LSTM is suitable for explainable machine learning.

G. REINFORCEMENT LEARNING (RL)

Reinforcement learning (RL) is a branch of machine learning, but unlike the commonly-known supervised learning, it is closer to human learning. Its exploration process is actually a process of gradually learning the rules of interaction through trials and responding to feedback from the environment. The RL model continuously communicates with the environment to find an optimal strategy through a series of attempts, and constantly adjusts its behavior based on the feedback. Fig. 8 provides the basic framework of reinforcement learning.

H. BOOSTING

Boosting is a learning model where multiple weak learners are combined to generate a strong classifier. Initially, a base weak classifier learns the training data. Next, in each iteration a weak learner is added to reduce the training error of previously applied weak learners. Freund and Schapire [7] proposed the first practical Boosting algorithm, AdaBoost. Gradient boosting algorithm proposed by Friedman *et al.* [8] is also widely applied for many optimization problems. In each iteration of the gradient boosting algorithm, the negative gradient of the current model on all samples is calculated. Next, a new weak classifier is trained with this value for adjusting the weight of the weak classifier. Finally, the model gets updated accordingly.

I. NAIVE BAYES (BAYES)

Naive Bayes Classifier is one of the simple probabilistic classifier based on Bayes theorem. This classifier has a strong assumption that the features are independent among themselves. From the training data, a likelihood probability is calculated for each feature. For an unknown input data, the posterior probability for each class is calculated using Bayes theorem. The class having maximum posterior probability value becomes the predicted level for the input data.

These ML techniques can be broadly divided into three categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. The ML techniques that require training labels are supervised learning. For example, support vector machine has been adopted in [9], [10] for supervised learning. While unsupervised learning does not use labels, but focus on extracting hidden features from input samples directly. The example for unsupervised learning is the 'clustering' algorithms [11], where definition of metric is the key for classifying [12]. Finally, reinforcement learning is a type of learning method that imitates human's learning process through continuous interaction and refinement. The detailed pros and cons are depicted in Table 1.

III. THREAT MODEL: HARDWARE SECURITY VULNERABILITIES

In this section, we first provide an overview of various SoC vulnerabilities. Next, we highlight the challenges associated analyzing SoC vulnerabilities.

A. OVERVIEW OF HARDWARE SECURITY VULNERABILITIES

SoC vulnerabilities and associated security threats can be broadly divided into the following categories.

1) MALICIOUS IMPLANTS (Hardware Trojans)

Hardware Trojan (HT) [13] is a malicious hardware modification that can leak secret information, degrade the performance of the system, or cause denial-of-service. It is a malicious modification of the target integrated circuit (IC) with two critical parts, trigger and payload. When the trigger is activated, the payload enables the malicious activity [14]. For example in Figure 9, when the output of the trigger logic is true, the output of the payload XOR gate will invert the expected output. The trigger is typically created using

TABLE 1. Advantages and disadvantages of various ML techniques for SoC security analysis.

ML Algorithms	Advantages	Disadvantages
Supervised Learning (e.g., SVM, DT, RF, LR)	(i) Easy to manipulate (ii) Clear definition of decision boundary (iii) Human-understandable training process (iv) Relatively less storage requirement (v) Knowing number of classes prior-training (vi) Better Transparency (vii) More accurate and reliable results	(i) Require golden designs or ICs; (ii) Easy to over-fitting; (iii)Improper to train large dataset
Unsupervised Learning (e.g., PCA, Clustering)	(i) Need no golden designs as references; (ii) Scalable and efficient for large designs; (iii) Good flexibility for various tasks.	(i) Unexpected model output; (ii) Easy to fall into local-optimization; (iii) Sensitive to noise; (iv) Sensitive to initialized condition
Reinforcement Learning	(i) Can solve very complex problem; (ii) Models can correct errors occurred during the training process by themselves; (iii) Good exploration & exploitation; (iv) Optimal solution where the only way for data collection is interacting with environment;	(i) Not preferable to solve simple problems; (ii) Need huge amount of data and, long training time; (iii) Based on Markovian assumption, which is not always held; (iv) Model performance heavily depends on design of reward function.

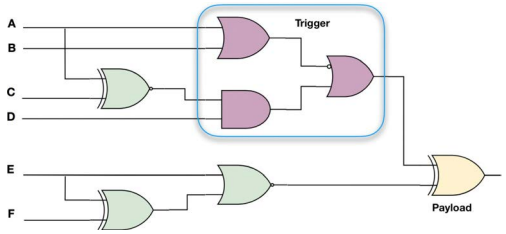

FIGURE 9. An example hardware Trojan constructed by a trigger logic (purple gates). Once the trigger condition is satisfied, the payload (yellow XOR gate) will invert the expected output. The gates of the original design are shown in green color.

a combination of rare events (such as rare signals or rare transitions) to stay hidden during normal execution. The payload represents the malicious impact HT will inflict to the target design, commonly resulting in information leakage or erroneous execution. HT can be applied in various levels including IP-level and bus-level [15]–[20]. A major challenge for Trojan identification is that Trojans are usually stealthy, as they are designed in a way that they can be activated under very rare conditions [21]. Due to this stealthy nature, it is infeasible to detect them using traditional functional validation methods. [22]

B. SUPPLY CHAIN VULNERABILITY (SCV)

Supply chain vulnerability can be broadly categorized into three fundamental threats: counterfeiting, overbuilding and recycling. IC counterfeiting is a serious problem that arises from the global semiconductor supply chain [23]. Some manufacturers use recycled ICs instead of genuine ones for reasons like out of stock or lower price. Counterfeit ICs with poor quality are usually overused, which degrade the quality of the manufactured products and lead to early product failures [24]. Counterfeiting of ICs has become a major challenge due to the difficulty of detection, and the lack of effective avoidance mechanisms [25]. Most of the existing approaches require costly rework and wasting a large amount of time. IC overbuilding is a bad situation where the foundry manufactures more ICs than required by the IC designer. Then during the flow of supply chain, an attacker with access to extra ICs can steal and claim ownership of the extra ICs and sell them illegally. More precisely, IC over-building attacks are launched by illegally copying or stealing authentic blueprints of SoC during the design, synthesis, or production phases, and result in illegal sales in the market, as demonstrated in [26].

C. REVERSE ENGINEERING (RE)

Reverse engineering of SoC [27], is an information invasion technique achieved by exploiting the backdoors in an SoC, either by faulty design or maliciously implantation [28]. A successful reverse engineering attack enables the adversary to uncover the IC design, extract its gate-level netlist to further infer its functionality [29]. This will lead to the revealing of inner details about the design, where the attacker can steal the intellectual property, or even improve it on their own product to achieve illegal advantage. The threat of reverse engineering is discussed in details in [30].

D. SIDE-CHANNEL LEAKAGE (SCL)

Side channel vulnerabilities arise from the fact that electronic devices inevitably produce physical emanation during execution, including but not limited to execution time, power consumption, path delay and electromagnetic emanation. These physical signatures can unintentionally reveal secret information from the device. For example, timing-attack is a typical approach to abuse the side-channel vulnerability, since timing information can be exploited to reveal memory information. Assuming that there is a private array in the memory with one pre-recorded entry in the cache, while the attacker wants to know the actual index of it. This can be achieved by traversing the entire array and measure the access time, the location with the shortest access time is the target one due to the huge difference of access time between cache and memory, as shown in Figure 10. This cache-based side-channel attack is widely applied in the famous Spectre and Meltdown attacks.

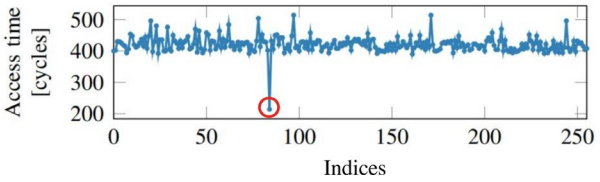

FIGURE 10. An example cache-based side channel attack. By traversing the entire array and recording the access time, the adversary can lock on the data corresponding to the index with the shortest access time, since it is most likely to be pre-stored in the cache.

E. CHALLENGES IN ANALYZING SOC VULNERABILITIES

While there are some successful attempts in protecting from these threats, there are various challenges in analysis and mitigation of SoC vulnerabilities. In this section, we briefly describe three important challenges: diversity of attacks, computation complexity, and real-time constraints.

1) DIVERSITY OF ATTACK SOURCES

The malicious parties trying to engage in IP security and privacy violation can be introduced throughout the IC design as well as the manufacturing process. For example, we illustrate three scenarios where an attacker can introduce vulnerabilities.

- Design Stage: Design of SoC starts with defining the general behavior and requirements using high-level languages. Attacks can occur through malicious modifications (e.g., inserting hardware Trojans, backdoor or leaky paths). Moreover, IP theft can also happen at the design stage.
- Synthesis Stage: The design is supposed to be synthesized to a gate-level netlist, where entities have write access to the netlist. They can inject malicious functionality in the design.
- Fabrication Stage: Due to the threat from untrusted fabrication foundries, it can lead to IP piracy as well as reverse engineering of the design to create counterfeit IPs. It can also lead to insertion of malicious implants (e.g., hardware Trojans).

Since sources of the vulnerability comes from various entities across the design cycle, it significantly increases the cost of designing comprehensive analysis and mitigation techniques.

F. COMPUTATIONAL COMPLEXITY

Another major challenge is high computation complexity. This comes in two different aspects. Most existing tasks tend to require brute-force methods for searching solutions. For example to generate test patterns for detecting hardware Trojans (discussed in Sec III), there is no golden rule to create efficient patterns. Therefore, many existing approaches generate new patterns by randomly flipping bits in a brute-force manner [31]. As a result, these approaches introduce high computation complexity and become impractical for large designs. Moreover, many analysis techniques boil down to SAT problems, which is NP-hard. Therefore, there is inherently no efficient implementation for these algorithms and researchers have to seek for approximation algorithms with a price of sacrificing accuracy.

1) VIOLATION OF REAL-TIME CONSTRAINTS

SoCs are also widely used in real-time systems such as cars, airplanes, and military and medical devices. The tasks in these systems need to finish before their deadlines. While minor violation in task deadlines may be tolerable in soft real-time systems, any violation of task deadlines can lead to catastrophic consequences in hard real-time systems. For real-time tasks, timing constraints enforce that the analyzing task should complete its execution before collected data expires. Similarly for protection techniques, their detection and recovering work should be completed before the end of the execution of malicious code. For example, once a ransomware attack has encrypted a lot of files already, it is too late to detect the ransomware attack. Unfortunately, existing approaches usually requires long run time and introduces considerable hardware complexity. In other words, these analysis and protection algorithms can be viewed as additional tasks in real-time systems. The additional computation and communication requirement can lead to violation of the real time constraints.

IV. SURVEY METHODOLOGY

To analyze diverse SoC vulnerabilities, there are a wide variety of machine learning (ML) based approaches. In this survey, we focus on some of the most commonly applied ML methods in the context of SoC security validation and verification. While there are a wide spectrum of hardware vulnerabilities, we have focused on hardware vulnerabilities that have ML-based detection methods in the literature. For example, destructive HT detection has been well studied [32], but out of the scope of this survey due to lack of ML-based detection approaches for destruction HT detection. Specifically, we analyze the following five types of techniques in the existing literature. The first two are dynamic approaches that rely on execution trace analysis. The last three are static approaches that rely on analysis of the specification.

- **Simulation-based Validation (SV):** Approaches aim on generating tests to activate malicious modifications and propagate the payload to observation points to check with the expected results from the golden design, or just to monitoring the packet diversity to detect abnormal situations [33]. Simulation is scalable but cannot provide any verification guarantees. The outcome depends on the quality of input tests and faces input space complexity.
- **Side-channel analysis (SCA):** As discussed in Section III, side-channel leakage can be abused by adversaries to reveal hidden information in an SoC. At the same time, side-channel signals can also be utilized to validate the security of a given design by physical signals like dynamic current [34]–[36], thermal [37], [38], and path-delay [39]. For example, designers can manually craft test vectors to trigger

the side channel signature emanation, by which they can evaluate the quality of SoC or detect potential vulnerabilities [40]. SCA faces uncertainty due to side-channel sensitivity associated with tiny Trojans.

- **Formal Verification (FV):** Formal verification is a commonly applied method in SoC validation. It uses formal mathematical methods to prove or disprove whether the system meets a certain specification or conforms to certain expected algorithm inferences. Formal verification provides mathematical guarantees about the trustworthiness of SoCs, but it can lead to state space explosion for large designs.
- **Printed Circuit Boards (PCB) Analysis:** PCB analysis is a specific method that inspects the components and solder joints of printed circuit boards to figure out damaged board components, staining corrosion, and possible presence of malicious implants. Specifically, PCB analysis includes measuring various plate thickness, identifying foreign materials, discovering defected parts, or creating the heatmap across the entire map. PCB analysis is valuable for detecting IC counterfeiting and hardware Trojans. PCB analysis has a natural compatibility with machine learning techniques since the commonly applied object detection techniques in computer vision domain can also be applied for PCB diagrams.
- **Heuristic Analysis (HA):** HA is a combinational analysis approach which consists of two major steps: feature selection and pattern recognition. *Feature selection* relies on extraction of specific features from the gate-level netlist based on either expert knowledge or heuristics. *Pattern recognition* analyzes various information of the extracted features to achieve the purpose of identification as well as classification of specific characteristics. As we can see, HA has a natural compatibility with machine learning approaches.

Simulation-based validation and Side-Channel Analysis are considered as **Dynamic Analysis** since they require execution of the designs, while formal verification, PCB analysis and heuristic analysis are classified as **Static Analysis**. Figure 11 shows the five classes of SoC vulnerability analysis approaches we consider in this survey. The subsequent sections discuss each class in detail. Each section introduces the recent publications in utilizing ML techniques for SoC vulnerability analysis. Related research efforts are compared and contrasted in terms of pros and cons. This survey mainly includes manuscripts published in the last 20 years.

V. SIMULATION-BASED VULNERABILITY ANALYSIS USING MACHINE LEARNING

Simulation-based validation (SV) approaches focus on test generation to activate hardware vulnerabilities. As shown in Figure 12, the presence of a malicious implant (hardware Trojan) can be detected if we can simulate with a suitable test vector. If the test vector activates the trigger of the Trojan, comparison of the simulation output with the expected output will reveal the presence of a Trojan in the implementation. Simulation-based vulnerability analysis starts with producing test vectors, which will be fed into the target design. By observing the outputs and comparing with the golden design, it can reveal important clues for possible vulnerabilities. Test generation is extremely important for both functional and trust validation of integrated circuits. It is a promising and general purpose approach, which possess good robustness against environmental noise or process variations. However, it may not be suitable for large-scale designs due to input-space complexity as well as prohibitive simulation cost.

In early days, random test generation was widely explored for simulation-based approaches due to its simplicity. However, there is no guarantee for activating stealthy Trojans using millions of random or constrained-random tests. MERO [31] proposed a statistical test generation scheme, which adopts the N-detect idea [41] to achieve better coverage. The heuristic behind is that if all rare signals are activated for at least N times, it is likely to activate the rare trigger conditions when N is sufficiently large. The left side of Figure 13 shows an overview of MERO. It starts with random test generation followed by a brute-force process of flipping bits to increase the number of rare values being satisfied. It provides promising result for small benchmarks, but it introduces long execution time and scalability concerns, making it unsuitable for large benchmarks [42].

To address these issues, Lyu *et al.* proposed TARMAC [42] as shown on the right side of Figure 13. Like MERO, TAR-MAC also starts with random simulation to identify rare signals in the netlist. Next, it maps the design to a satisfiability graph, and converts the problem of satisfiability into a clique cover problem, where the authors use an SMT solver [43] to generate test patterns for each maximal clique. Although TARMAC performs significantly better than MERO in evaluated benchmarks, its performance is very unstable. This is due to the fact that TARMAC relies on random clique sampling, making its performance dependent on the quality of sampled cliques.

Let us take a closer look at these approaches. There are two major problems that affect the performance of existing efforts: rareness heuristic and test generation complexity.

A. WEAKNESS OF RARENESS HEURISTIC

Existing methods rely on rareness heuristic for activating HT triggers. However, in [44], the author rigorously discussed the inconsistency between rare nodes and trigger nodes. According to their experimental evaluation, rare nodes are not necessarily trigger nodes, and vice versa. Reliance on rareness hurts the genuine nodes with rare attribute (e.g., low switching activity). More-over, a smart implementation of HT can exploit the mixture of both rare nodes and genuine (non-rare) nodes to obfuscate Trojan detection.

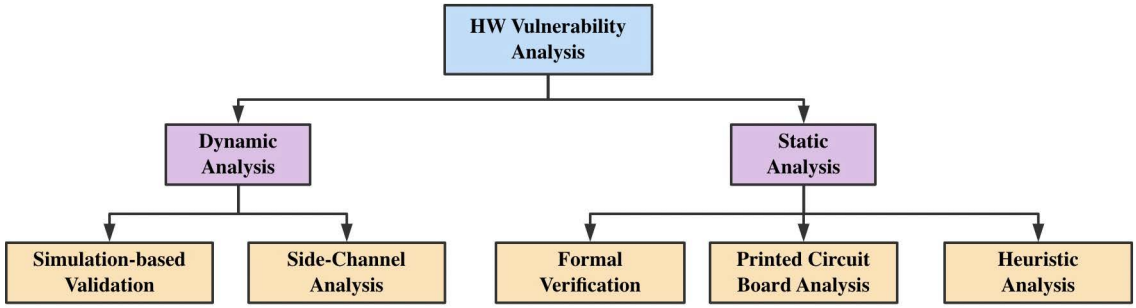

FIGURE 11. Four classes of ML-based hardware vulnerability analysis discussed in this survey.

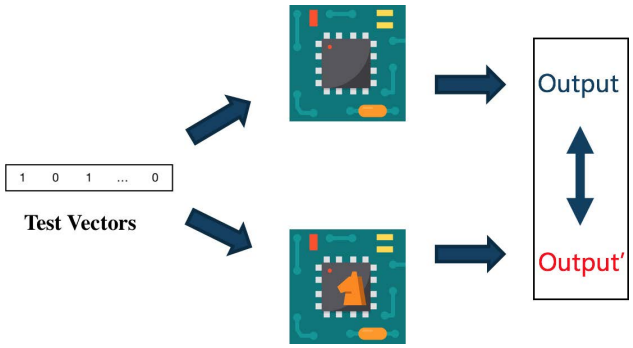

FIGURE 12. An overview of simulation-based validation of security vulnerabilities.

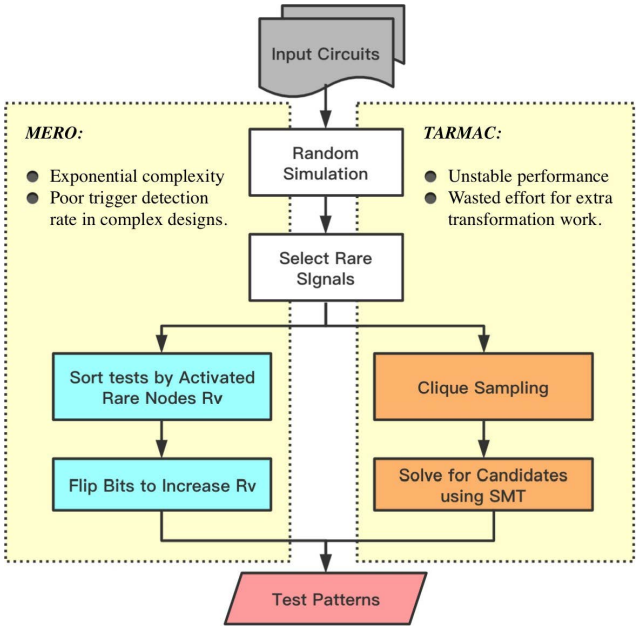

FIGURE 13. Overview of state-of-the-art logic testing techniques: MERO [31] and TARMAC [42].

B. TEST GENERATION COMPLEXITY

Another major drawback of existing approaches is high computation complexity. Existing efforts ignores the interaction between intermediate test vectors and circuit that typically provides useful feed-back. For example, if a newly generated test vector significantly decreases the number of triggered rare nodes, then the current parameters of the test generation algorithm needs to get adjusted to avoid wasted effort (time). While this intuition is likely to help in guiding the test generation process, it is ignored by both MERO and TARMAC. MERO generates new test patterns by blindly flipping bits in a brute-force manner using random strategy, and TARMAC performs random sampling of cliques without taking the feedback into consideration. In [45], the authors also observed this problem and proposed a genetic algorithm [46] based approach. However, their evaluation shows that they require even longer test generation time. This is due to the combined effects of time-consuming training and slow convergence of genetic algorithm in the later stages of evolution.

Actually, an ideal test generation algorithm should satisfy these two crucial requirements to address the presented challenges.

- **Test Effectiveness:** Exploiting not only the rareness, but also the testability of signals to improve trigger coverage.
- **Test Generation Efficiency:** Efficiently making use of feedback in intermediate steps to save test generation time.

These requirements motivate researchers to seek help from machine learning. We discuss promising ML-based approaches for hardware Trojan (HT) detection as well as side-channel leakage (SCL) analysis.

1) ML-BASED DETECTION OF HARDWARE TROJANS USING SIMULATION-BASED VALIDATION

Several approaches have been proposed to use machine learning to detect hardware Trojans. Most of these approaches work by applying different tests to maximize the probability to activate Trojan's trigger conditions to detect untrustworthy behavior of the design. In these techniques, a machine is modeled and trained to work as the generator for test vectors, whose outputs are collected and compared to a trusted region in a multidimensional space.

Hasegawa *et al.* [47] proposed a static Trojan detection technique using SVM-based classification of gate-level netlists. This method manually extracts several features for any potentially suspicious candidate S in a gate-level netlist to differentiate a Trojan-inserted netlist from a safe one. The features are constructed based on the stealthy behavior of

hardware Trojans. An SVM classifier is trained on these features to detect unknown Trojans. A similar runtime approach has been proposed for many-core platforms [48]. These approaches can achieve high accuracy (80%) in detecting Trojans. However, they also have a high false positive rate as they might mark many benign components as suspicious. The accuracy of such approaches has been improved by Pan *et al.* [49] utilizing improved feature selection and use of reinforcement learning models to reduce false positive rate.

2) ML-BASED SIDE-CHANNEL LEAKAGE ANALYSIS UTILIZING SIMULATION-BASED VALIDATION

Due to the extraordinary potential in solving complex searching and classification tasks, ML techniques have been widely adopted for side-channel leakage (SCL) analysis. Hospodar *et al.* [50] presented the first study on applying ML in side-channel analysis. Afterwards, due to the variance of different side channel signals and the difficulty of protecting against environmental noise, some approaches mainly target on specific benchmarks or limited type of side channel attacks. For example, Alam *et al.* proposed a ML-based safeguard against micro-architectural side-channel attacks [51]. This paper emphasized the importance of using time-series data for training ML models in security domain to correlate the execution trace with the hidden information in sequential data flow. In [52], a SVM-based detector targeting cache-based side-channel attacks (SCA) was proposed and achieve a 96.57% accuracy on average. These techniques provide promising results, but they have usability restrictions.

In [53], the author systematically discussed various cases of SCA detection at runtime using a wide variety of ML models. They provide insights on various aspects of performance for gate-level designs, while similar techniques are necessary for other abstraction levels. To address this limitation, Wang *et al.* proposed a multi-phase ML framework for detection and identification of cache-based side-channel attacks [54].

A electronic system-level approach was discussed in [55]. As shown in Figure 14, the framework utilizes two test generation schemes: fixed by human expert knowledge and random generation. Next, it selects side-channel leakage window and applies a ML-based clustering algorithm to plot the distribution of side-channel information. Finally, a metric was induced to distinguish malicious design from golden designs.

3) OVERHEAD AND EFFECTIVENESS OF SURVEYED APPROACHES

Table 2 shows a summary of the papers we have surveyed related to ML-based vulnerability analysis using simulation-based validation.

We use two metrics (detection accuracy and hardware overhead) to enable relative comparison between existing methods. We use a three-step scale (low, average, high) to indicate the vulnerability detection accuracy. Specifically, we use 'high', 'average', and 'low' when the vulnerability detection accuracy is >95%, 90% 95%, and <90%, respectively. Similarly, we use a three-step scale (low, average, high) to indicate the hardware overhead in terms of area and power requirements compared to the original design (without detection method). Note that some of the related efforts provided only power overhead while others provided power/energy values. We use 'high', 'average', and 'low' when the vulnerability detection accuracy is >10%, 5% 10%, and <5%, respectively. Ideally, a designer would like to use a solution that has 'high' detection accuracy and 'low' hardware overhead. The second and third columns outline the threat models and corresponding ML models, respectively. The last two columns provide the overhead as well as effectiveness, respectively.

VI. ML-BASED FORMAL VERIFICATION OF SECURITY VULNERABILITIES

Formal verification (FV) is promising in hardware validation as they evaluate the functionality and security of the design using discrete mathematical models [56]. In general, FV process is accomplished by first abstracting the system, generating a discrete mathematical model, and then providing formal proofs for certain requirements, as shown in Figure 15.

There are a variety of formal verification methods including model (property) checking, satisfiability (SAT) solving [57], equivalence checking, theorem proving, etc. We consider the following three categories in this survey due to the fact that existing ML-based approaches applied one of these methods as assistance tool to either perform strategy optimization or work as a key step during the formal verification of hardware vulnerabilities.

- **Satisfiability (SAT) Solving:** The first step in Boolean SAT is to convert the SoC design into a Boolean proposition formula such as conjunctive normal form (CNF). A SAT solver is applied to obtain the Boolean assignments of input variables so that the value of formula is true. When such an assignment exists, the formula is considered to be *satisfiable*, and vice versa. Many of the functional and security validation problems in SoC domain can be mapped to SAT problems.
- **Model Checking (MC):** MC focuses on abstracting SoC's execution into a finite-state machine (FSM) and the properties of interest are expressed in temporal logic. MC then exhaustively explores the state space to check if the implementation (FSM) satisfies the given properties. In case a property is not satisfied (potential vulnerability), MC generates a counterexample that can be analyzed to fix the vulnerability.
- **Equivalence Checking (EC):** EC can be used to formally prove that different representations of a design display the same functionality — nothing more, nothing less. Therefore, it has a natural fit to detect any vulnerabilities assuming that the golden model is available.

The remainder of this section describes how these formal verification techniques are used by the existing ML-based

TABLE 2. Threat model, defense, overhead and effectiveness of work in existing literature related to ML-based SV approaches. Threat model: Types of threats in SoC to deal with, including Hardware trojan (HT) and Side-channel leakage (SCL). ML model: Machine learning models applied. Specifically, we use "∗" to indicate multiple(>5) ML models applied. Hardware Overhead: We use a three-step scale (low, average, high) to indicate the overhead in terms of area and power requirements. Detection Accuracy: We use a three-step scale (low, average, high) to indicate the vulnerability detection accuracy.

	Threat Model	ML Model	Hardware Overhead	Detection Accuracy
Hasegawa, 2017 [47]	HT	SVM	Average	Low
Kulkarni, 2016 [48]	HT	SVM	Average	Average
Zhixin, 2020 [49]	HT	RL	Average	High
Hospodar, 2014 [50]	SCL	*	High	Low
Alam, 2017 [51]	SCL	RF, SVM, Bayes	Average	Average
Tong, 2020 [52]	SCL	SVM	Average	Low
Mushtaq, 2018 [53]	SCL	*	Low	Low
Hussain, 2018 [54]	SCL	DNNs	Average	Average
Zhang, 2020 [55]	SCL	SVM , DNNs	High	High

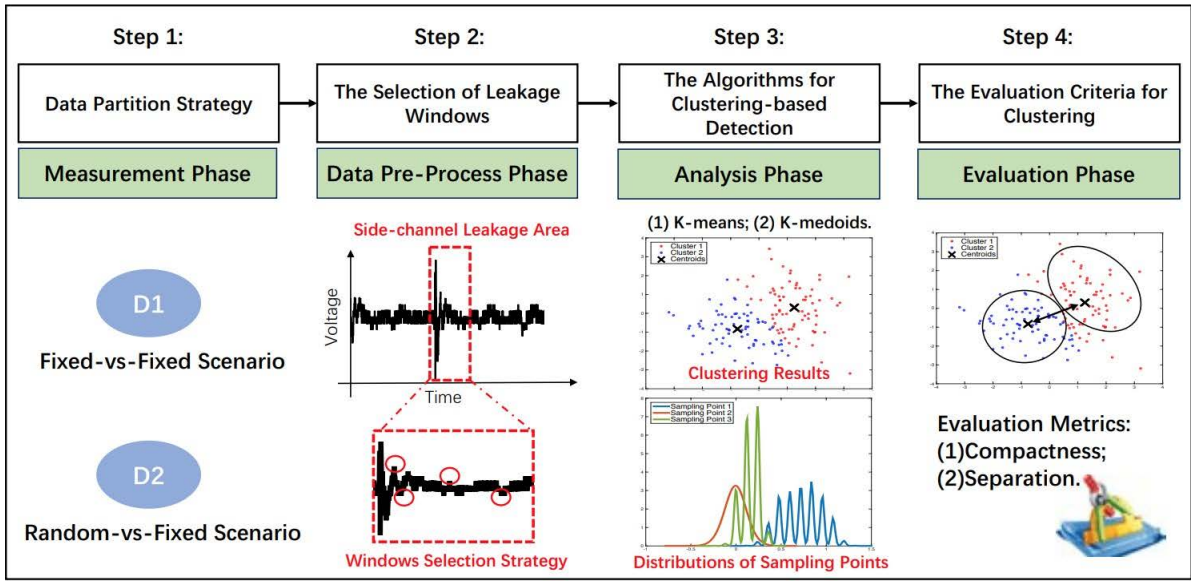

FIGURE 14. The basic steps for performing ML-based electronic system-Level SCA detection in [55].

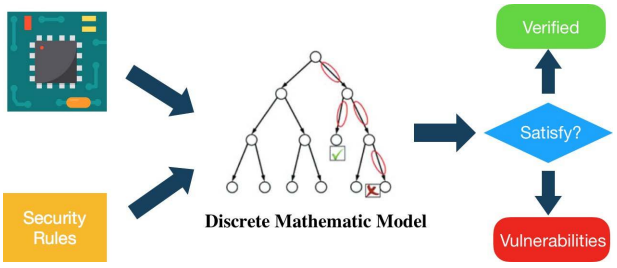

FIGURE 15. An overview of formal verification of security vulnerabilities.

hardware vulnerability analysis. Notice content in this section mainly discuss methods utilizing ML to assist formal verification in general, which can be further extended to tackle hardware security verification. which is a broad new area remaining unexplored.

A. ML-BASED SATISFIABILITY SOLVING FOR VULNERABILITY ANALYSIS

The difficulty of SAT problems comes from the fact that it is hard to determine the environmental settings, timing for various tasks, and the selection of branching variables. Therefore, ML techniques are usually applied to develop approximation algorithms, or work as an optimizer for tasks such as dynamic determination of the most feasible solvers. To predict the runtime of a SAT solver for better program switching in case the current one takes too long, Horvitz *et al.* proposed a Bayesian approach to tackle the computational cost [58]. This inspires people to apply ML in other scenarios. For example, for solving as many instances as possible within a given time budget, selecting the best restart strategy on various design features represents a classification ML task.

Haim & Walsh [59] proposed a smart selection of the restart strategy, where features are chosen based on portfolio-based algorithm.

Selection of the branching variable is another major concern. Choosing the most appropriate branching variable is crucial for improving solvers' runtime by minimizing back-tracking. This makes it a perfect match for a reinforcement learning problem. Imaging along a SAT solver's working process, it chooses the next variable to branch on, a reward value can be attached to the variable choice as score, so that it maximizes the progress for solving the instance. Liang *et al.* [60], [61] explored two different reward

computations based on different branching heuristics: conflict-based heuristic for variable selection [60], and learning rate branching [61]. Both heuristics rewarded the generation of learned clauses locally to save time. Fröhlich *et al.* [62] proposed another framework, where they penalize the candidate variable choices to minimize the number of unsatisfied clauses. Lagoudakis and Littman [63], penalize branching rules instead to obtain a better strategy, but its solving time is too long, which reduces its usability.

ML techniques can even be applied to determine the best solving algorithm. It is a well-known fact that certain families of instances can be better solved with specific algorithms. This could be seen as a regression ML problem, when the goal is to predict an algorithm's continuous probability of success. SATzilla [64] proposed one of the popular solvers, which allows to switch to another algorithm when the current one takes too long inducing low-efficiency. Similarly, AutoFolio [65] selects algorithms based on features similar to the ones in SATzilla.

Several research efforts focus on applying ML to help configuring SAT solvers' parameters. Hutter, Hoos and Leyton-Brown [66] defined SMAC (Sequential Model-based Algorithm Configuration), a technique to generalize the classical optimization algorithm by using training, based mostly on SATzilla's features, resulting in a highly parametrized framework.

Most ML techniques for SAT focus on some assistance work, since tackling the whole SAT problem through ML is difficult. In reality, it has shown some promising research directions. For example, Wu *et al.*'s work predicts 3-CNF instance satisfiability with seven features from given design, where they reuse the partial prediction to determine which value is preferable for a branching literal [67]. However, this approach cannot be directly applied to other class of instances since the number of clauses and variables is at equilibrium particularly for 3-CNF instances, making instances neither underconstrained nor overconstrained. Devlin and O'Sullivan [68] as well as Xu *et al.* [69] studied several classifiers for predicting the other types of instances, where they tried to minimize the number of necessary features to build simple and elegant classifiers, but the entire framework is not tested through large-scale benchmarks.

B. ML-BASED MODEL CHECKING FOR VULNERABILITY ANALYSIS

Model checking analysis utilizes a finite-state machine (FSM) based representation of the implementation to validate properties of interest, or generate counterexamples triggering vulnerabilities in the SoC design. These characteristics make it unsuitable with machine learning methods, therefore, there are no ML-related works for model checking. Instead, many contributions focus on helping the speed of detecting counterexamples, or reducing false positives during model checking.

Araragi and Cho [70] targeted the production of counterexamples using reinforcement learning. The authors kept track of the premise occurrence at the state space level, and rewarded explorations that stayed on paths with possible contradiction that can generate counterexamples. However, this method would lead to cyclic, or very long paths of execution. Behjati *et al.* [71] follow a similar approach, where they developed a reinforcement learning framework. The agent in their algorithm is punished when following non-accepting cycles, and rewarded when finding unfair accepting cycles, which could possibly lead to the property's invalidation. Clarke *et al.* [72] implemented an linear programming based FSM-refinement technique for model-checking deployed on real hardware circuits. They train a ML model to automatically extract spurious counterexamples (false alarms).

Researchers are inspired to develop ML-based approaches for extracting the most common counterexamples for a given design (i.e., error patterns). This is closely related to the ML task of extracting frequent items and pattern recognition. In [74], Pira *et al.* characterized error patterns as specific sequence of transitions in FSM, and by applying a graph-based model with a variation of the APriori algorithm. These patterns are discovered on smaller systems with similar architectural design, therefore, they can be used to guide model checking on larger systems. In this way, the cost for MC is significantly reduced.

C. ML-BASED EQUIVALENCE CHECKING FOR VULNERABILITY ANALYSIS

Hu *et al.* proposed a equivalence checking method based on ML [75]. The work aims at performing equivalence checking between register-transfer level (RTL) design and system-level description. They transform both type of designs into finite state machines with datapath (FSMD) and compare the path pairs to measure the equivalence. It recognizes the corresponding path pairs of the model using a SVM model, which avoids a blind path selection work in general equivalence checking. Their method can dramatically reduce the time complexity. Their subsequent work [76] checks more ML models to show trade-off between effectiveness and efficiency. Also, in [77], the author proposed an alternative ML-based equivalence checking, where they first convert RTL design into dataflow graphs, then apply a graph-to-sequence neural network model to achieve equivalence checking. The proposed method achieves an accuracy of 96% with promising time efficiency.

For some of surveyed works, the authors did not provide any analysis about overhead. Entries presented in Table are our best estimations based on ML models utilized in their works.

D. OVERHEAD AND EFFECTIVENESS OF SURVEYED APPROACHES

Table 3 shows a summary of the papers we have surveyed related to ML-based vulnerability analysis using formal verification.

VII. ML-BASED HEURISTIC ANALYSIS FOR DETECTION OF VULNERABILITIES

As discussed in Sec IV, Heuristic Analysis (HA) is a combinational analysis approach by first performing feature extraction followed by pattern recognition. There is a natural compatibility of HA with a wide variety of machine learning approaches. Most of the existing works are designed for specific purposes, and rely on expert knowledge and experience to select important features from the designs. In other words, ML-based solutions discussed in this section are utilized as a tool in the final step, classification, whereas the crucial step (feature selection) is performed manually by a designer with deep understanding of design features and its security implications.

In [78], the authors target integrated circuit (IC) aging analysis by proposing a two-stage method for detecting recycled FPGAs. Both stages rely on machine learning via support vector machines (SVM) for classification. In the first stage, they compare the frequencies of ring oscillators (ROs) distributed on the FPGAs against a golden model as the crucial feature to distinguish recycled and fresh FPGA. This is an initialization step, where they raise red flags for suspicious components in the entire circuit. While in the second phase, they performed a short aging step on the suspect FPGAs and exploits the aging speed reduction (due to prior usage) to confirm the cases marked by the first step. According to their experimental evaluation, their approach can distinguish fresh and recycled FPGAs with 100% accuracy for a variety of benchmarks.

Huang *et al.* [79] presented an approach to detect counterfeit ICs using SVM. Instead of ROs, they first observe the distribution of process variation across devices. Next, they build a simple parametric measurement to map this circuit feature into a 2-D space. Finally, an SVM classifier performs the job of counterfeit detection, which is shown in Figure 17. However, these approaches work well when we know the proper feature to be fed into the ML models. In other words, promising solutions for a specific scenario (e.g., countermeasure against IC overfeiting) cannot be utilized for other types of SoC vulnerabilities.

To apply HA for HT detection, Hasegawa in [80], developed a Trojan-feature extraction algorithm at gate-level, and utilize it to perform hardware-Trojan detection using SVM classifier. This work was also extended to similar framework with different ML models including random forest (RF) [73] and neural networks [47]. In general, they focus on hardware-Trojan detection using machine learning in IC design step. They outlined 11 most effective features that reflect Trojan's impact on netlists, as shown in Figure 16. By using RF as the classifier, they obtained 100% true positive rate and 100% true negative rate in several Trust-HUB benchmarks.

In [81], heuristic analysis is performed by signal correlation. It estimates the statistical correlation between the signals in a design, and explores how this estimation can be used in a clustering algorithm to detect the Trojan logic. Since it is based on heuristic of using statistical model to estimate the signal correlations of gate-level netlist, it neither need the circuit to be brought to the triggering state, nor the effect of the Trojan payload to be propagated and observed at the output. This idea was inherited in [82], where the authors directly apply the transition probability as the feature for analysis. To address the order sensitivity problem, a stacked long short-term memory network is designed to build a robust HT detection model. Following this assumption, a more sophisticated work was proposed in [83]. Lu *et al.* extracted information entropy instead of signal correlation from circuits as critical features. This work is built based on one assumption that to maintain stealthy concealment, Trojans should be inserted in regions with low controllability and observability, which will result in low transition probability of Trojan logic.

Dong *et al.* [84] analyzed the existing Trojan-net features from [80] to propose five new hardware-Trojan features. To further develop time-efficient approaches, they utilized the gradient boosting algorithm to train the ML model for HT detection [84]. Their algorithm applied the scoring mechanism of the eXtreme Gradient Boosting (XGBoost) [85] to set up a new feature set. Their experimental results demonstrate that they are able to obtain an average F-score of 87.75%. Similar ideas are also explored in [86] and [87], where gradient boosting algorithm was applied on features extracted from RTL level and gate-level, respectively. In [88], the ML models' performance is further enhanced by hyperparameter tuning of RF model. In [89], HT detection uses class weighted XGBoost, where higher weights are assigned to minority Trojan-inserted class to remove the need for oversampling, which improves the efficiency of HT detection.

The above mentioned works focus on features that are either static signal features [73], [78] or reaction features [79].

Zhou *et al.* [90] combined HA and ML techniques for HT detection, where they analyzed the structural features of IP cores and HT triggers in gate-level circuits. Their method is specifically designed for detecting HTs triggered by less toggled signals. They abstract the circuit into a graph, and extract special structures that are commonly adopted by HT implants. After obtaining the general pattern of those suspicious structures, they apply ML model for the classification work, and their experimental results yields a 100% detection rate. However, these algorithms share the same bottleneck as the above methods that require golden reference model since ML models applied here are all supervised learning. To address this limitation, Xue *et al.* [11] proposed a hybrid clustering model to achieve golden-Free HT detection. They achieve this by utilizing two attack models to imitate untrustworthy parties. Then through adversarial data generation, they produce the pool of adversarial samples of possible HTs, from which features of HTs were extracted. By utilizing hybrid clustering ensemble method, they were able to classify if a given circuit contains malicious implants.

Trojan feature	Description
fan_in_x	The number of logic-gate fanins up to x-level away from the net n.
in_flipflop_x	The number of flip-flops up to x-level away from the input side of the net n.
out_flipflop_x	The number of flip-flops up to x-level away from the output side of the net n.
in_multiplexer_x	The number of multiplexers up to x-level away from the input side of the net n.
out_multiplexer_x	The number of multiplexers up to x-level away from the output side of the net n.
in_loop_x	The number of up to x-level loops.
out_loop_x	The number of up to x-level loops.
in_const_x	The number of constants up to x-level away from the input side of the net n.
out_const_x	The number of constants up to x-level away from the output side of the net n.
in_nearest_pin	The minimum level to the primary input from the net n.
out_nearest_pout	The minimum level to the primary output from the net n.
{in, out}_nearest_flipflop	The minimum level to any flip-flop from the input or output side of the net n.
{in, out}_nearest_multiplexer	The minimum level to any multiplexer from the input or output side of the net n.

FIGURE 16. The 11 most effective HT features to help ML detection. [73].

TABLE 3. Threat model, defense, overhead and effectiveness of work in existing literature related to ML-based formal verification approaches. Threat model: types of threats in SoC to deal with, including hardware Trojan (HT), supply-chain vulnerability (SCV) and reverse engineering (RE). ML model: machine learning models applied. Hardware overhead: we use a three-step scale (low, average, and high) to indicate the overhead in terms of area and power requirements. Detection accuracy: We use a three-step scale (low, average, and high) to indicate the vulnerability detection accuracy.

	Threat Model	ML Model	Hardware Overhead	Detection Accuracy
Horvitz, 2013 [58]	HT	Naive Bayesian	Low	Average
Haim, 2009 [59]	HT	SVM	Average	Average
Liang, 2016 [60], [61]	SCV	RL	High	High
Frohlich, 2015 [62]	SCV	RL	High	High
Lindauer, 2015 [65]	RE	LR	Low	Low
Araragi, 2006 [70]	HT	RL	High	Average
Behjati, 2009 [71]	HT	RL	High	Average
Clarke, 2018 [72]	SCV	LR	Average	Average
Pira, 2016 [74]	HT	DNNs	High	High
Hu, 2020 [75], [76]	HT	DNNs	High	Average
Kommrusch, 2020 [77]	HT	DNNs	High	High

TABLE 4. Threat model, defense, overhead and effectiveness of work in existing literature related to ML-based heuristic analysis approaches. Threat model: types of threats in SoC to deal with, including hardware Trojan (HT) and supply-chain vulnerability (SCV). ML model: machine learning models applied. Hardware overhead: we use a three-step scale (low, average, and high) to indicate the overhead in terms of area and power requirements. Detection accuracy: we use a three-step scale (low, average, and high) to indicate the vulnerability detection accuracy.

	Threat Model	ML Model	Hardware Overhead	Detection Accuracy
Dogan, 2014 [78]	SCV	SVM	Low	Average
Huang, 2012 [79]	SCV	SVM	Average	High
Hasegawa, 2016 [80]	HT	SVM	Low	Low
Hasegawa, 2017 [47], [73]	HT	RF, DNNs	High	High
Cakir, 2015 [81]	HT	Clustering	Low	Low
Lu, 2019 [83]	HT	RNN	High	Average
Lu, 2021 [82]	HT	Clustering	Average	Average
Zhou, 2016 [90]	HT	LR	Low	Average
Xue, 2019 [11]	HT	DNNs	High	High
Dong, 2019 [84]	HT	Boosting	Low	High
Reddy, 2022 [88]	HT	RF & Boosting	Low	High
Sharma, 2020 [89]	HT	Boosting	Low	High

We summarize the overhead and effectiveness of surveyed papers related to ML-based heuristic analysis in Table 4.

VIII. ML-BASED SIDE-CHANNEL ANALYSIS OF HARDWARE VULNERABILITIES

The field of side-channel analysis (SCA) has gained significant attention for hardware vulnerability analysis [91]. SCA has now been practiced in design companies and test laboratories, and the security of products against side-channel attacks has been significantly improved over the years. However, there are still many unresolved weaknesses remained making SCA less effective. In general, SCA consists of two steps: execution using suitable input (test) patterns and comparison with the expected (golden) side-channel signatures.

Security experts create test vectors to trigger side-channel signatures, which can evaluate the quality of the SoC or detect potential vulnerabilities. Although both SCA and simulation-based validation (SV) relies on test generation and comparison with golden design, there are few fundamental differences.

- Unlike SV which relies on functional values at output ports, SCA relies on side-channel signatures such power consumption/dissipation, path delay, electro-magnetic emanation, etc.

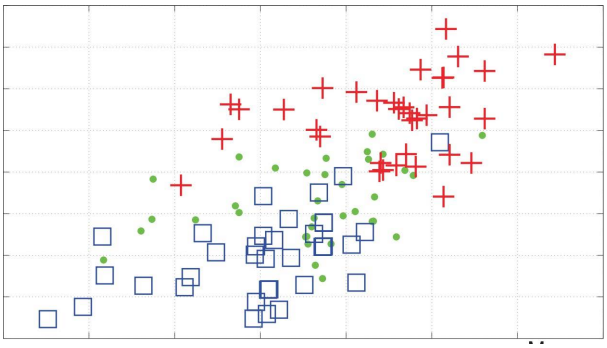

FIGURE 17. In [79], IC with different aging level are mapped in 2D space, where the classification work is done by using SVM to distinguish data points in 2D space.

- Detection of tiny Trojans is difficult using SCA, since the side-channel footprint of the tiny Trojan can easily hide in environmental noise or manufacturing process margins. This is not a problem for SV-if the Trojan can be activated, there will be mismatch in the expected outputs (assuming the Trojan alters the functionality).
- It is hard to activate Trojan trigger during SV, since it is infeasible to simulate all possible input (test) patterns in a large design (exponential number of test vectors). This is not a problem in SCA, since it does not require activation of the Trojan trigger.

SCA is widely adopted in scenarios where the model is poorly known, since the task can be approximated to a profiling phase. SCA is typically used for hardware Trojan detection in integrated circuits by analyzing various side-channel signatures, such as timing, power and path delay. An overview of side-channel attacks and countermeasures is presented in Figure 18.

Hospodar *et al.* [92] proposed one of the early efforts to deal with the application of ML techniques for SCA. They used a variant of SVM called least square support vector machine (LS-SVM) to distinguish power traces of an unprotected software AES implementation regarding three properties of the S-Box output to detect implanted HT. However, their selection of features are based on expertise, and may not be applicable in other scenarios. The most relevant features were discovered by a Sum Of Squared pairwise T-differences (SOST) analysis in [93] using Pearson correlation and Principal Component Analysis (PCA). They observed that the choice of the LS-SVM parameters significantly affects the performance of the classification, whereas the size of the training set is less important.

Heuser and Zohner used multi-class SVM classification to make assumptions about the intermediate value byte of an AES implementation running on an ATMega-256-1 microcontroller [94]. They show that their SVM model is more stable since it relaxes the assumption that the data underlie a multivariate Gaussian distribution. This provided the basis for the work of Bartkewitz and Lemke-Rust [95], who designed probabilistic multi-class SVMs the same way. Next, they applied a technique called normal-based feature selection. Here, the absolute values of the weight parameters were carefully refined to determine if a corresponding feature has significant influence on the classification performance. Small weight values are therefore set to zero to disregard unimportant features. This idea actually follows the explainable machine learning approaches, where the model provides an importance ranking of features to outline the most crucial features of SoC design for distinguishing it from golden designs. The efficiency of the approach was measured through a wide variety of benchmarks. They observed that SVM-based model is perfect if the security problem can be directly transferred into a linear classification problem, whereas the RBF kernel is appropriate for non-linear problems if using kernel tricks as discussed in Sec II.

Banciu *et al.* investigated several ML-based classifiers in the context of single trace attacks [96]. These type of attacks assume an adversary which only has access to a single attack trace. When targeting symmetric ciphers, the attacks should be error tolerant in a sense that side-channel leakage information for an intermediate value can be a set of possible values. Examples from literature are discussed in [97], where a set of assumptions are pre-defined to guarantee their method's correctness. In the study of [98], different ML models are evaluated, where Bayesian, SVM, DNN, DT and RF were trained to produce a ranked list of hardware features with given power consumption traces obtained from an AES implementation. In their experimental evaluation, only Bayesian, RF and SVM performed well across the two data sets and different numbers of traces/features used for training.

Picek *et al.* presented an approach called hierarchical classification [99]. The idea is to explore the natural clustering of the leakage in order to arrange the sensitive variables in a tree structure. It starts by dividing all collected traces in the corresponding platform by their sensitivity to environmental noise, and then each subset is classified into different sensitive variables. The main purpose of their work is to give a rank of sensitive regions inside the circuit whose signals are likely to emanate side-channel information about the entire circuit. Naive Bayes, DT, RF and SVM were considered as classification algorithms and evaluated along with standard template attacks. In most cases, SVM model performed the best. The authors also proposed to combine the hierarchical approach with a standard DNN classifier to increase the accuracy.

Picek *et al.* showed in an another study the importance of proper parameter tuning when using ML techniques for side-channel analysis [101]. From the set of examined supervised classifiers (SVM, RF and Adaboost), the best results in terms of classification accuracy under cross-validation through parameter tuning were obtained for SVM. However, RF and Adaboost performed only slightly worse with their optimal settings, but were far more robust to parameter value changes. It is furthermore shown that a carefully tuned algorithm is able to reach a relatively high accuracy (more than 70% in the worst case) even if only a small number of relevant features is used.

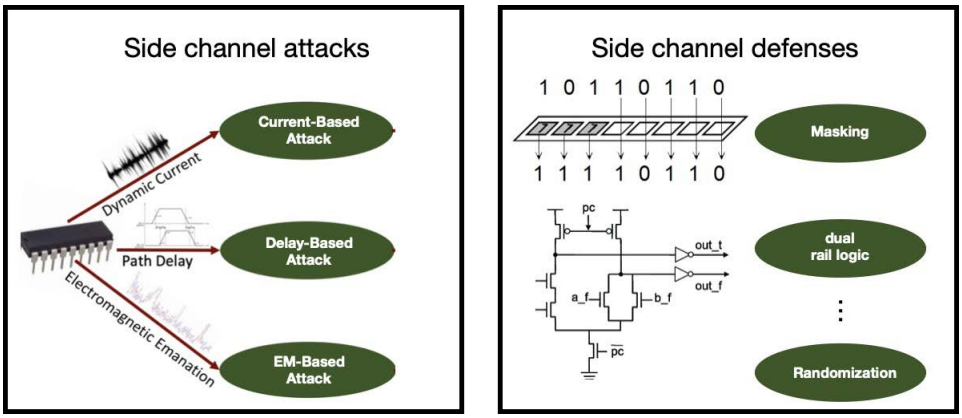

FIGURE 18. A brief overview of commonly applied side channel attacks and defences.

Most research works in ML-based SCA focus on selection of the features, and tuning of parameters. These two major parts were usually done with the information gain method, such as C4.5 [102] algorithm, which is applied to grow DT. The authors also presented a novel side-channel metric called data confusion factor that quantifies the difficulty of a given ML problem regarding a certain data set.

There are recent research efforts that try to combine the advantages of simulation-based validation with SCA. The basic idea is to maximize side-channel sensitivity by generating efficient test patterns that can maximize activity in the suspicious regions while minimize activity in the rest of the design. For example, in [38], the author propose an efficient test generation technique to detect Trojans using delay-based side channel analysis, which is an automated test generation algorithm to produce test patterns that are likely to activate trigger conditions, and change critical paths. However, it is not effective due to two fundamental limitations: (i) the difference in path delay between the golden design and Trojan inserted design is negligible compared with environmental noise and process variations and (ii) it relies on manually crafted rules for test generation, and require a large number of simulations, making it impractical for industrial designs.

To address these challenges, Pan *et al.* [100] proposed a test generation method using reinforcement learning for delay-based Trojan detection. This approach utilizes critical path analysis to generate test vectors that can maximize the side-channel sensitivity. Experimental results demonstrate that this method can significantly improve both side-channel sensitivity (59% on average) and test generation time (17× on average) compared to state-of-the-art test generation techniques. The framework of this algorithm is depicted in Figure 19. Another golden-free Trojan detection method was proposed in [103] that utilizes Trojan Trigger's EM side-channel fingerprint. The method is based on the unsupervised clustering analysis of golden RTL code instead of actual fabricated chip. In other words, it eliminates the need for a trusted golden fabricated chip as a reference. This method achieves > 99% detection accuracy.

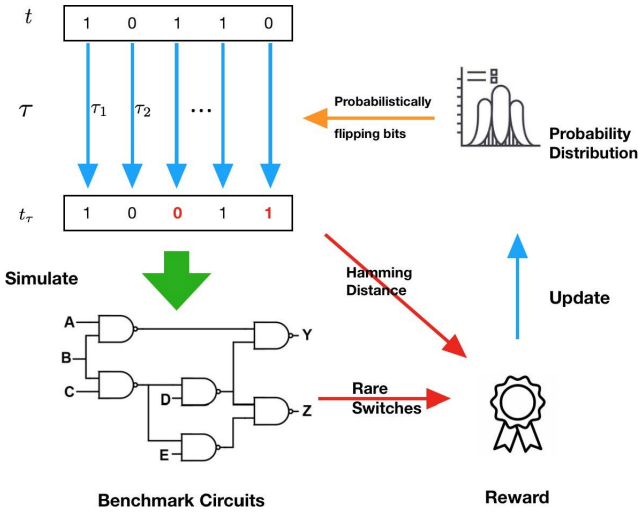

FIGURE 19. Illustration of the stochastic reinforcement learning method in [100].

Table 5 summarizes the papers we have surveyed related to ML-based vulnerability analysis using side-channel analysis.

IX. ML-BASED PCB ANALYSIS OF HARDWARE VULNERABILITIES

A printed circuit board (PCB) contains conductive pathways to connect different parts of the board, which is utilized to mechanically support and electrically connect electronic components. The pathways are inscribed onto the board according to the architecture of the PCB design. By analyzing the layout and connectivity from PCB, users are able to detect potential security threats such as malicious implanting or IC counterfeiting. However, PCBs are prone to a variety of defects including, but not limited to, improper manufacturing, scratching, and destructive delayering during the analysis procedure. These defects severely degrade the performance of traditional PCB-level analysis. Several illustrative examples of PCB from [104] are shown in Figure 20.

As we can see, PCB-level analysis has a natural compatibility with ML-based approaches. ML-based computer vision tasks include image target classification and target

TABLE 5. Threat model, defense, overhead and effectiveness of work in existing literature related to ML-based SCA approaches. Threat model: Types of threats in SoC to deal with, including Hardware Trojan (HT) and Supply-chain vulnerability (SCV). ML model: Machine learning models applied. Specifically, we use "∗" to indicate that it involves multiple ML models. Hardware overhead: We use a three-step scale (low, average, high) to indicate the overhead in terms of area and power requirements. Detection accuracy: We use a three-step scale (low, average, high) to indicate the vulnerability detection accuracy.

	Threat Model	ML Model	Hardware Overhead	Detection Accuracy
Hospodar, 2011 [93]	HT	SVM	Low	Low
Mangard, 2010 [94]	HT	SVM	Low	Low
Prouff, 2012 [95]	HT	SVM	Average	Average
Bartkewitz, 2014 [96]	HT	Multi-Class SVM	High	Average
Banciu, 2015 [97]	SCV	*	Average	Average
Renauld, 2009 [99]	HT	*	High	Average
Picek, 2017 [100]	HT	Clustering	Low	Average
Nowroz, 2014 [38]	HT	*	Low	Low
Pan, 2020 [101]	HT	RL	High	High
He, 2020 [105]	HT	Clustering	Low	High

TABLE 6. Threat model, defense, overhead and effectiveness of work in existing literature related to ML-based PCB analysis approaches. Threat model: types of threats in SoC to deal with, including hardware Trojan (HT) and supply-chain vulnerability (SCV). ML model: machine learning models applied. Specifically, we use "∗" to indicate that it involves multiple ML models. Hardware overhead: we use a three-step scale (low, average, and high) to indicate the overhead in terms of area and power requirements. Detection Accuracy: We use a three-step scale (low, average, and high) to indicate the vulnerability detection accuracy.

	Threat Model	ML Model	Hardware Overhead	Detection Accuracy
Cheong, 2019 [108]	HT	CNN	High	High
Yuk, 2018 [106]	HT	RF	Low	High
Zakaria, 2020 [109]	SCV	DNN	Average	Average
Hu, 2020 [110]	SCV	CNN	High	Average
Domenic, 2020 [111]	SCV	*	Average	Average
Zhang, 2019 [112]	SCV	SVM	High	Average
Mallaiyan, 2021 [115]	HT&SCV	*	Low	Average

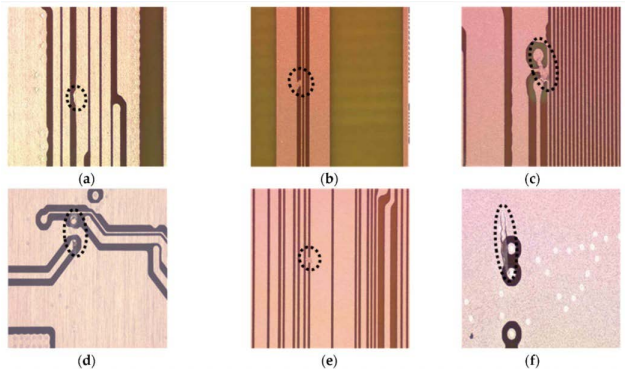

FIGURE 20. Illustrative examples of printed circuit boards (PCB) from [104].

localization, which is particularly suitable for PCB structure analysis and component recognition. As a result, researchers start to seek nondestructive ML-based methods to assist PCB-level analysis. In general, a deep neural network can be trained for PCB component detection to identify malicious, counterfeit, reused, or recycled components.

As a data-driven technique, ML requires huge amount of samples to train the ML model. Gayathri *et al.* introduce PCB-METAL [105], a printed circuit board (PCB) high resolution image dataset, which can be utilized for computer vision and ML-based component analysis. The dataset consists of 984 high resolution images of 123 unique PCBs with bounding box annotations for ICs, capacitors, resistors, and inductors. The dataset is useful for image-based PCB analysis such as component detection, PCB classification, circuit design extraction, etc. A hardware Trojan detection approach based on components recognition in PCB using convolutional neural network (CNN) is proposed in [106]. It achieved 99% accuracy with the capability of recognizing up to 25 different components in the board. Yuk *et al.* proposed another feature-learning-based PCB inspection method via speeded-up Random Forest in [104]. Similarly, for defect detection, Zakaria *et al.* proposed an automated detection of PCB defects by using machine learning in electronic manufacturing [107]. Hu *et al.* proposed a surface defect detection method with faster CNN and feature pyramid network [108].

For supply chain vulnerability, Stern *et al.* [109] utilize the fingerprint information of microprocessors based on electromagnetic emanation (EM) from three vendors. They utilized both unsupervised (PCA) and supervised (LR) learning on all ICs to successfully determine their vendors. Statistical analysis and machine learning techniques are used to achieve high classification accuracy. A novel authorization methodology to prevent IC counterfeit through supply chain is illustrated in [110]. In this work, a ring oscillator (RO) array is inserted into the selected IC. The counterfeit PCB/equipment can be detected by judging whether the periods of ROs in the resonant and non-resonant state of the PCB matches the database from the authentic ones. The proposed authorization structure has been implemented on some authentic and counterfeit FPGA development boards, and more than 90% counterfeit PCBs can be identified by a 2-class SVM classifier.

ML-based PCB analysis approaches are also widely applied to introduce supply chain vulnerabilities, especially for reverse engineering. In [111], Navid *et al.* introduce a

nondestructive approach for PCB reverse engineering based on X-ray tomography, with low associated time and cost. In [112], Erozan *et al.* presented a robust reverse engineering methodology based on supervised machine learning, starting from image acquisition all the way to netlist extraction.

The authors in [113] summarized the pros and cons of utilizing ML-based approaches in PCB level. They examined various ML methods applied in computer vision domain on PCB analysis. They also summarized the challenges of applying machine learning for PCB component recognition and how to address it. Specifically, ML-based PCB analysis suffers from variance, evolving scope, and the need for dataset. In [113], the authors proposed a specialized network called the electronic component localization and detection network (ECLAD-Net) to address these challenges.

Table 6 summarizes the papers we have surveyed related to ML-based vulnerability analysis using side-channel analysis.

X. CONCLUSION AND FUTURE DIRECTIONS

System security relies on the security of each of the components including hardware, firmware and software. Hardware security is critical since some of the software security guarantees are based on the hardware-root-of-trust. Machine learning is very promising in efficient detection and mitigation of hardware vulnerabilities. In this survey, we have provided a comprehensive review of hardware vulnerability analysis using machine learning techniques. We have discussed how existing approaches utilized machine learning techniques for hardware security verification using simulation-based validation, formal verification, heuristic analysis as well as side-channel analysis.

While the existing approaches have shown promising results, there are many related problems that need to be solved in order to design secure and trustworthy systems. In this section, we outline few research directions that we believe are interesting and open problems in the domain.

A. ADVERSARIAL MACHINE LEARNING

While machine learning has been widely used in defending systems agains malicious attacks, machine learning models also impose security risks. For example, machine learning can also be used by adversaries to launch attacks. In most of the reviewed papers, accuracy is less than 100%, which means that there are still in correctly classified data points in the test set. When machine learning is applied in realistic scenarios, misclassification of a single attack is enough to exploit the whole system. More-over, attackers can mislead classification process by injecting attacks that are deliberately classified as safe while they are in fact malicious. While machine learning is a promising approach for hardware security, if the detection accuracy and the true positive rate are not 100%, the system is under severe security risk. When machine learning models are deployed in security related applications, only the detected attacks are taken into consideration. However, instances that are marked as normal should be treated as suspicious, i.e., these instances may be misclassified as normal while they are in fact attacks.

B. SIDE-CHANNEL SENSITIVITY

Though machine learning has been widely adopted in SCA, existing side-channel analysis techniques have two major bottlenecks: (i) they are not suitable in detecting Trojans in real-world applications since the difference between the golden design and a Trojan inserted design is usually negligible compared to environmental noise and process variations, and (ii) they are not effective in creating robust signatures due to their reliance on random and ATPG based test patterns. Therefore, there is a lot of scope for developing efficient test generation techniques for improving side-channel sensitivity.

C. FORMAL VERIFICATION TECHNIQUES

In Section VI, we described machine learning techniques to address formal verification in general. However, they are not especially designed for hardware security verification. This inherently brings two attractive research directions. First is to directly apply these ML-assisted formal verification techniques to tackle practical hardware security verification. Second is to develop specific ML methods that especially designed for verification. in future research works, which remains an unexplored land in security domain.

D. EXPLAINABLE MACHINE LEARNING

Typically machine learning algorithms are utilized as a tool in certain steps of HW security analysis instead of the entire process. One of the major reasons that makes pure ML-based verification hard is that the implicit models (e.g. neural networks) are difficult to grasp and understand for humans. They act like a black box to produce desired outputs without explanation. The ML models are completely nontransparent since there is no proper indication on what kind of mechanism contributes to promising results. This brings two challenges in applying ML in real applications. (i) Specifying which models to deploy is difficult, since there is no criteria, and human cannot explain why certain model performs well for specific tasks. Most of the existing works address this by exploring a wide spectrum of different models to select a beneficial ML model. (ii) Limited information for detection results. From a nontransparent ML model, end-users merely obtain the final detection result without interpretation. Therefore, they are unable to locate the threats or perform scene clearing operations, and it is difficult for them to take suitable precautions. There is a need for utilizing explainable machine learning for hardware vulnerability analysis.

The continuous emergence of new techniques and novel methods can be exploited by an adversary to implement more intelligent attacks. While effort should be devoted to solving the remaining outstanding issues efficiently at low-cost [114], emerging threats are also a major concern. For example, the vulnerabilities discussed in this paper are related to integrated

circuits designed using VLSI (CMOS) technology. The security landscape may be drastically different if we consider hardware designed using various emerging technologies. Therefore, the security researchers will always have a lot of items in their plate for understanding emerging vulnerabilities and design efficient machine learning techniques to detect and mitigate future vulnerabilities.

REFERENCES

[1] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[2] (2021). *Common Weakness Enumeration*. [Online]. Available: https://cwe.mitre.org/

[3] G. Sumathi, L. Srivani, D. T. Murthy, K. Madhusoodanan, and S. A. V. S. Murty, "A review on HT attacks in PLD and ASIC designs with potential defence solutions," *IETE Tech. Rev.*, vol. 35, no. 1, pp. 64–77, 2018.

[4] J. Zhang and G. Qu, "Recent attacks and defenses on FPGA-based systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 3, pp. 1–24, 2019.

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

[6] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[7] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[8] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[9] P. Gaikwad, J. Cruz, P. Chakraborty, S. Bhunia, and T. Hoque, "Third-party hardware IP assurance against trojans through supervised learning and post-processing," 2021, *arXiv:2111.14956*.

[10] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A massively parallel FPGA-based coprocessor for support vector machines," in *Proc. 17th IEEE Symp. Field Program. Custom Comput. Mach.*, Apr. 2009, pp. 115–122.

[11] M. Xue, R. Bian, W. Liu, and J. Wang, "Defeating untrustworthy testing parties: A novel hybrid clustering ensemble based golden models-free hardware trojan detection method," *IEEE Access*, vol. 7, pp. 5124–5140, 2019.

[12] Q. Cui, K. Sun, S. Wang, L. Zhang, and D. Li, "Hardware trojan detection based on cluster analysis of Mahalanobis distance," in *Proc. 8th Int. Conf. Intell. Hum.-Mach. Syst. Cybern. (IHMSC)*, vol. 1, Aug. 2016, pp. 234–238.

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

[14] J. Francq and F. Frick, "Introduction to hardware Trojan detection methods," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 770–775.

[15] X. Wang, Y. Zheng, A. Basak, and S. Bhunia, "IIPS: Infrastructure IP for secure SoC design," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2226–2238, Aug. 2015.

[16] T. Boraten and A. Karanth Kodi, "Packet security with path sensitization for NoCs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 1136–1139.

[17] N. Fern, I. San, C. K. Koc, and K.-T.-T. Cheng, "Hiding hardware trojan communication channels in partially specified SoC bus functionality," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 9, pp. 1435–1444, Sep. 2017.

[18] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proc. 53rd ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[19] N. Fern, I. San, Ç. Kaya Koç, and K.-T. (Tim) Cheng, "Hardware trojans in incompletely specified on-chip bus systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2016, pp. 527–530.

[20] S. K. Haider, C. Jin, M. Ahmad, D. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 1, pp. 18–32, Feb. 2019.

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

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

[23] Y. Jin, D. Maliuk, and Y. Makris, "A post-deployment IC trust evaluation architecture," in *Proc. IEEE 19th Int. On-Line Test. Symp. (IOLTS)*, Jul. 2013, pp. 224–225.

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

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

[26] A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security assurance for system-on-chip designs with untrusted IPs," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 7, pp. 1515–1528, Jul. 2017.

[27] M. Fyrbiak, S. Strauss, C. Kison, S. Wallat, M. Elson, N. Rummel, and C. Paar, "Hardware reverse engineering: Overview and open challenges," in *Proc. IEEE 2nd Int. Verification Secur. Workshop (IVSW)*, Jul. 2017, pp. 88–94.

[28] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *Proc. 21st Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2016, pp. 655–660.

[29] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," in *Proc. 48th Design Autom. Conf. (DAC)*, 2011, pp. 333–338.

[30] A. Alaql, S. Chattopadhyay, P. Chakraborty, T. Hoque, and S. Bhunia, "LeGO: A learning-guided obfuscation framework for hardware IP protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 4, pp. 854–867, Apr. 2022.

[31] R. Chakraborty, F. Wolff, and S. Paul, "MERO: A statistical approach for hardware trojan detection," in *Proc. CHES*, 2009, pp. 396–410.

[32] A. Saini, G. Kundra, and S. Kalra, "A survey on hardware trojan detection: Alternatives to destructive reverse engineering," in *Proc. 2nd Int. Conf. Comput., Commun., Cyber-Secur.* Springer, 2021, pp. 885–897.

[33] C. Liu, P. Cronin, and C. Yang, "A mutual auditing framework to protect IoT against hardware trojans," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 69–74.

[34] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic, "Detecting trojans through leakage current analysis using multiple supply pad I_{DDQ}s," *IEEE Trans. Inf. Forensics Security*, vol. 5, no. 4, pp. 893–904, Dec. 2010.

[35] T. Wehbe, V. J. Mooney, A. Q. Javaid, and O. T. Inan, "A novel physiological features-assisted architecture for rapidly distinguishing health problems from hardware Trojan attacks and errors in medical devices," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 106–109.

[36] B. Shanyour and S. Tragoudas, "Detection of low power trojans in standard cell designs using built-in current sensors," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[37] D. Forte, C. Bao, and A. Srivastava, "Temperature tracking: An innovative run-time approach for hardware Trojan detection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2013, pp. 532–539.

[38] A. N. Nowroz, K. Hu, F. Koushanfar, and S. Reda, "Novel techniques for high-sensitivity hardware trojan detection using thermal and power maps," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 33, no. 12, pp. 1792–1805, Dec. 2014.

[39] F. N. Esirci and A. A. Bayrakci, "Hardware Trojan detection based on correlated path delays in defiance of variations with spatial correlations," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 163–168.

[40] F. Koushanfar and A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits Trojan detection," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 1, pp. 162–174, Mar. 2011.

[41] I. Pomeranz and S. M. Reddy, "A measure of quality for n-detection test sets," *IEEE Trans. Comput.*, vol. 53, no. 11, pp. 1497–1503, Nov. 2004.

[42] Y. Lyu and P. Mishra, "Automated trigger activation by repeated maximal clique sampling," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 482–487.

[43] L. Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. TACAS*, 2008, pp. 337–340.
[44] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 2, pp. 338–350, Feb. 2017.
[45] M. A. Nourian, M. Fazeli, and D. Hely, "Hardware trojan detection using an advised genetic algorithm based logic testing," *J. Electron. Test.*, vol. 34, no. 4, pp. 461–470, Aug. 2018.
[46] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, U.K.: Oxford Univ. Press, 1996.
[47] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks," in *Proc. IEEE 23rd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2017, pp. 227–232.
[48] A. Kulkarni, Y. Pino, and T. Mohsenin, "SVM-based real-time hardware trojan detection for many-core platform," in *Proc. 17th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2016, pp. 362–367.
[49] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Proc. 26th Asia South Pacific Design Autom. Conf.*, Jan. 2021, pp. 408–413.
[50] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: A first study," *J. Cryptograph. Eng.*, vol. 1, no. 4, p. 293, 2011.
[51] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks," IACR Cryptol. ePrint Arch., Tech. Rep. 2017/564, 2017, p. 564.
[52] Z. Tong, Z. Zhu, Z. Wang, L. Wang, Y. Zhang, and Y. Liu, "Cache side-channel attacks detection based on machine learning," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec./Jan. 2020, pp. 919–926.
[53] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, "Machine learning for security: The case of side-channel attack detection at run-time," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 485–488.
[54] H. Wang, H. Sayadi, G. Kolhe, A. Sasan, S. Rafatirad, and H. Homayoun, "Phased-guard: Multi-phase machine learning framework for detection and identification of zero-day microarchitectural side-channel attacks," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 648–655.
[55] L. Zhang, D. Mu, W. Hu, and Y. Tai, "Machine-learning-based side-channel leakage detection in electronic system-level synthesis," *IEEE Netw.*, vol. 34, no. 3, pp. 44–49, May/Jun. 2020.
[56] N. Fern and K.-T. Cheng, "Pre-silicon formal verification of JTAG instruction opcodes for security," in *Proc. IEEE Int. Test Conf. (ITC)*, Oct./Nov. 2018, pp. 1–9.
[57] W. Gong and X. Zhou, "A survey of SAT solver," in *Proc. AIP Conf.*, 2017, vol. 1836, no. 1, Art. no. 020059.
[58] E. J. Horvitz, Y. Ruan, C. P. Gomes, H. Kautz, B. Selman, and D. Maxwell Chickering, "A Bayesian approach to tackling hard computational problems," 2013, *arXiv:1301.2279*.
[59] S. Haim and T. Walsh, "Restart strategy selection using machine learning techniques," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.* Springer, 2009, pp. 312–325.
[60] J. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Exponential recency weighted average branching heuristic for SAT solvers," in *Proc. AAAI Conf. Artif. Intell.*, 2016, vol. 30, no. 1, pp. 3434–3440.
[61] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.* Springer, 2016, pp. 123–140.
[62] F. Fröhlich, P. Thomas, A. Kazeroonian, F. J. Theis, R. Grima, and J. Hasenauer, "Inference for stochastic chemical kinetics using moment equations and system size expansion," *PLOS Comput. Biol.*, vol. 12, no. 7, Jul. 2016, Art. no. e1005030.
[63] M. G. Lagoudakis and M. L. Littman, "Learning to select branching rules in the DPLL procedure for satisfiability," *Electron. Notes Discrete Math.*, vol. 9, pp. 344–359, Jun. 2001.
[64] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, Jul. 2008.
[65] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *J. Artif. Intell. Res.*, vol. 53, pp. 745–778, Aug. 2015.
[66] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown, "Performance prediction and automated tuning of randomized and parametric algorithms: An initial investigation," in *Proc. AAAI Workshop Learn. Search: Schedule*, 2006, pp. 1–7.
[67] H. Wu, "Improving SAT-solving with machine learning," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, Mar. 2017, pp. 787–788.
[68] D. Devlin and B. O'Sullivan, "Satisfiability as a classification problem," in *Proc. 19th Irish Conf. Artif. Intell. Cogn. Sci.*, 2008.
[69] L. Xu, H. Hoos, and K. Leyton-Brown, "Predicting satisfiability at the phase transition," in *Proc. AAAI Conf. Artif. Intell.*, 2012, vol. 26, no. 1, pp. 1–7.
[70] T. Araragi and S. M. Cho, "Checking liveness properties of concurrent systems by reinforcement learning," in *Proc. Int. Workshop Model Checking Artif. Intell.* Springer, 2006, pp. 84–94.
[71] R. Behjati, M. Sirjani, and M. N. Ahmadabadi, "Bounded rational search for on-the-fly model checking of LTL properties," in *Proc. Int. Conf. Fundam. Softw. Eng.* Springer, 2009, pp. 292–307.
[72] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, *Handbook of Model Checking*, vol. 10. Springer, 2018.
[73] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.
[74] E. Pira, V. Rafe, and A. Nikanjam, "EMCDM: Efficient model checking by data mining for verification of complex software systems specified through architectural styles," *Appl. Soft Comput.*, vol. 49, pp. 1185–1201, Dec. 2016.
[75] J. Hu, T. Li, and S. Li, "Equivalence checking between SLM and RTL using machine learning techniques," in *Proc. 17th Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2016, pp. 129–134.
[76] J. Hu, Y. Hu, Q. Lv, W. Wang, G. Wang, G. Chen, K. Wang, Y. Kang, and H. Yang, "A path-based equivalence checking method between system level and RTL descriptions using machine learning," *J. Circuits, Syst. Comput.*, vol. 30, no. 4, Mar. 2021, Art. no. 2150074.
[77] S. Kommrusch, T. Barollet, and L.-N. Pouchet, "Equivalence of dataflow graphs via rewrite rules using a Graph-to-Sequence neural model," 2020, *arXiv:2002.06799*.
[78] H. Dogan, D. Forte, and M. M. Tehranipoor, "Aging analysis for recycled FPGA detection," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2014, pp. 171–176.
[79] K. Huang, J. M. Carulli, and Y. Makris, "Parametric counterfeit IC detection via support vector machines," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Oct. 2012, pp. 7–12.
[80] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists based on machine learning," in *Proc. IEEE 22nd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2016, pp. 203–206.
[81] B. Çakir and S. Malik, "Hardware trojan detection for gate-level ICs using signal correlation based clustering," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 471–476.
[82] R. Lu, H. Shen, Z. Feng, H. Li, W. Zhao, and X. Li, "HTDet: A clustering method using information entropy for hardware Trojan detection," *Tsinghua Sci. Technol.*, vol. 26, no. 1, pp. 48–61, 2021.
[83] R. Lu, H. Shen, Y. Su, H. Li, and X. Li, "GramsDet: Hardware trojan detection based on recurrent neural network," in *Proc. IEEE 28th Asian Test Symp. (ATS)*, Dec. 2019, pp. 111–1115.
[84] C. Dong, J. Chen, W. Guo, and J. Zou, "A machine-learning-based hardware-Trojan detection approach for chips in the Internet of Things," *Int. J. Distrib. Sensor Netw.*, vol. 15, no. 12, 2019, Art. no. 1550147719888098.
[85] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2016, pp. 785–794, doi: 10.1145/2939672.2939785.
[86] T. Han, Y. Wang, and P. Liu, "Hardware trojans detection at register transfer level based on machine learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.
[87] 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 *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2021, pp. 412–417.

[88] M. N. Reddy, M. Latchmana Kumar, P. B. S. Kumar, S. Thirumalai, and M. Nirmala Devi, "Performance enhancement by tuning hyperparameters of random forest classifier for hardware Trojan detection," in *Proc. Int. Conf. Data Sci. Appl.* vol. 2022. Springer, 2022, pp. 177–191.
[89] R. Sharma, N. K. Valivati, G. K. Sharma, and M. Pattanaik, "A new hardware trojan detection technique using class weighted XGBoost classifier," in *Proc. 24th Int. Symp. VLSI Design Test (VDAT)*, Jul. 2020, pp. 1–6.
[90] E.-R. Zhou, S.-Q. Li, J.-H. Chen, L. Ni, Z.-X. Zhao, and J. Li, "A novel detection method for hardware trojan in third party IP cores," in *Proc. Int. Conf. Inf. Syst. Artif. Intell. (ISAI)*, Jun. 2016, pp. 528–532.
[91] Y. Liu, K. Huang, and Y. Makris, "Hardware trojan detection through golden chip-free statistical side-channel fingerprinting," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6.
[92] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: A first study," *J. Cryptograph. Eng.*, vol. 1, no. 4, p. 293, 2011.
[93] S. Mangard and F.-X. Standaert, *Cryptographic Hardware and Embedded Systems: CHES 2010*. Springer, 2010.
[94] E. Prouff, *Constructive Side-Channel Analysis and Secure Design*. Springer, 2014.
[95] T. Bartkewitz and K. Lemke-Rust, "Efficient template attacks based on probabilistic multi-class support vector machines," in *Proc. Int. Conf. Smart Card Res. Adv. Appl.* Springer, 2012, pp. 263–276.
[96] V. Banciu, E. Oswald, and C. Whitnall, "Reliable information extraction for single trace attacks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 133–138.
[97] R. Spreitzer, G. Palfinger, and S. Mangard, "SCAnDroid: Automated side-channel analysis of Android APIs," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2018, pp. 224–235.
[98] M. Renauld and F.-X. Standaert, "Algebraic side-channel attacks," in *Proc. Int. Conf. Inf. Secur. Cryptol.* Springer, 2009, pp. 393–410.
[99] S. Picek, A. Heuser, A. Jovic, and A. Legay, "Climbing down the hierarchy: Hierarchical classification for machine learning side-channel attacks," in *Proc. Int. Conf. Cryptol. Afr.* Springer, 2017, pp. 61–78.
[100] Z. Pan, J. Sheldon, and P. Mishra, "Test generation using reinforcement learning for delay-based side-channel analysis," in *Proc. ICCAD*, Nov. 2020, pp. 1–7.
[101] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-channel analysis and machine learning: A practical perspective," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 4095–4102.
[102] A. A. Aldino and H. Sulistiani, "Decision TREE C4. 5 algorithm for tuition aid grant program classification (case study: Department of information system, Universitas Teknokrat Indonesia)," *Jurnal Ilmiah Edutic: Pendidikan dan Informatika*, vol. 7, no. 1, pp. 40–50, 2020.
[103] J. He, H. Ma, Y. Liu, and Y. Zhao, "Golden chip-free trojan detection leveraging Trojan trigger's side-channel fingerprinting," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–18, 2020.
[104] E. H. Yuk, S. H. Park, C.-S. Park, and J.-G. Baek, "Feature-learning-based printed circuit board inspection via speeded-up robust features and random forest," *Appl. Sci.*, vol. 8, no. 6, p. 932, 2018.
[105] G. Mahalingam, K. M. Gay, and K. Ricanek, "PCB-METAL: A PCB image dataset for advanced computer vision machine learning component analysis," in *Proc. 16th Int. Conf. Mach. Vis. Appl. (MVA)*, May 2019, pp. 1–5.
[106] L. K. Cheong, S. A. Suandi, and S. Rahman, "Defects and components recognition in printed circuit boards using convolutional neural network," in *Proc. 10th Int. Conf. Robot., Vis., Signal Process. Power Appl.* Springer, 2019, pp. 75–81.
[107] S. Zakaria, A. Amir, N. Yaakob, and S. Nazemi, "Automated detection of printed circuit boards (PCB) defects by using machine learning in electronic manufacturing: Current approaches," in *Proc. IOP Conf., Mater. Sci. Eng.*, 2020, vol. 767, no. 1, Art. no. 012064.
[108] B. Hu and J. Wang, "Detection of PCB surface defects with improved faster-RCNN and feature pyramid network," *IEEE Access*, vol. 8, pp. 108335–108345, 2020.
[109] A. Stern, U. Botero, F. Rahman, D. Forte, and M. Tehranipoor, "EMFORCED: EM-based fingerprinting framework for remarked and cloned counterfeit IC detection using machine learning classification," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 363–375, Feb. 2020.
[110] D. Zhang, Y. Han, and Q. Ren, "A novel authorization methodology to prevent counterfeit PCB/Equipment through supply chain," in *Proc. IEEE 4th Int. Conf. Integr. Circuits Microsyst. (ICICM)*, Oct. 2019, pp. 128–132.
[111] N. Asadizanjani, M. Tehranipoor, and D. Forte, "PCB reverse engineering using nondestructive X-ray tomography and advanced image processing," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 7, no. 2, pp. 292–299, Feb. 2017.
[112] A. T. Erozan, M. Hefenbrock, M. Beigl, J. Aghassi-Hagmann, and M. B. Tahoori, "Reverse engineering of printed electronics circuits: From imaging to netlist extraction," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 475–486, 2019.
[113] M. A. Mallaiyan Sathiaseelan, O. P. Paradis, S. Taheri, and N. Asadizanjani, "Why is deep learning challenging for printed circuit board (PCB) component recognition and how can we address it?" *Cryptography*, vol. 5, no. 1, p. 9, Mar. 2021.
[114] A. Sengupta, S. Bhadauria, and S. P. Mohanty, "TL-HLS: Methodology for low cost hardware trojan security aware scheduling with optimal loop unrolling factor during high level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 4, pp. 655–668, Apr. 2017.

ZHIXIN PAN (Member, IEEE) received the B.E. degree from the Department of Software Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Science and Engineering, University of Florida. His research interests include hardware security and trust, malware detection, post-silicon debug, data mining, and machine learning.

PRABHAT MISHRA (Fellow, IEEE) received the Ph.D. degree in computer science from the University of California at Irvine, in 2004. He is a Professor with the Department of Computer and Information Science and Engineering and a UF Research Foundation Professor with the University of Florida. His research interests include embedded and cyber-physical systems, hardware security and trust, energy-aware computing, systems-on-chip validation, machine learning, and quantum computing. He is a Distinguished Scientist of ACM. He currently serves as an Associate Editor for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and *ACM Transactions on Embedded Computing Systems*.