Hardware Trojan Detection Using Machine Learning: A Tutorial

KEVIN IMMANUEL GUBBI, BANAFSHEH SABER LATIBARI, ANIRUDH SRIKANTH, and TYLER SHEAVES, University of California, Davis, USA
SAYED ARASH BEHESHTI-SHIRAZI and SAI MANOJ PD, George Mason University, USA
SATAREH RAFATIRAD, AVESTA SASAN, and HOUMAN HOMAYOUN, University of California, Davis, USA
SOHEIL SALEHI, University of Arizona, USA

With the growth and globalization of IC design and development, there is an increase in the number of Designers and Design houses. As setting up a fabrication facility may easily cost upwards of $20 billion, costs for advanced nodes may be even greater. IC design houses that cannot produce their chips in-house have no option but to use external foundries that are often in other countries. Establishing trust with these external foundries can be a challenge, and these foundries are assumed to be untrusted. The use of these untrusted foundries in the global semiconductor supply chain has raised concerns about the security of the fabricated ICs targeted for sensitive applications. One of these security threats is the adversarial infestation of fabricated ICs with a **Hardware Trojan (HT)**. An HT can be broadly described as a malicious modification to a circuit to control, modify, disable, or monitor its logic. Conventional VLSI manufacturing tests and verification methods fail to detect HT due to the different and un-modeled nature of these malicious modifications. Current state-of-the-art HT detection methods utilize statistical analysis of various side-channel information collected from ICs, such as power analysis, power supply transient analysis, regional supply current analysis, temperature analysis, wireless transmission power analysis, and delay analysis. To detect HTs, most methods require a Trojan-free reference golden IC. A signature from these golden ICs is extracted and used to detect ICs with HTs. However, access to a golden IC is not always feasible. Thus, a mechanism for HT detection is sought that does not require the golden IC. **Machine Learning (ML)** approaches have emerged to be extremely useful in helping eliminate the need for a golden IC. Recent works on utilizing ML for HT detection have been shown to be promising in achieving this goal. Thus, in this tutorial, we will explain utilizing ML as a solution to the challenge of HT detection. Additionally, we will describe the **Electronic Design Automation (EDA)** tool flow for automating ML-assisted HT detection. Moreover, to further discuss the benefits of ML-assisted HT detection solutions, we will demonstrate a **Neural Network (NN)**-assisted timing profiling method for HT detection. Finally, we will discuss the shortcomings and open challenges of ML-assisted HT detection methods.

CCS Concepts: • **Security and privacy** → **Malicious design modifications;**

This work was partly supported by the National Science Foundation through Computing Research Association for CIFellows #2030859.

Authors' addresses: K. I. Gubbi, B. Saber Latibari, A. Srikanth, T. Sheaves, S. Rafatirad, A. Sasan, and H. Homayoun, University of California, Davis, Davis, California, USA; S. A. Beheshti-Shirazi, George Mason University, Fairfax, VA, USA; S. M. PD, George Mason University, Fairfax, VA, USA; S. Salehi, University of Arizona, Tucson, Arizona, USA.

1539-9087/2023/04-ART46 $15.00
https://doi.org/10.1145/3579823

Additional Key Words and Phrases: Hardware security, hardware Trojan detection, side-channel analysis, machine learning, neural networks

ACM Reference format:
Kevin Immanuel Gubbi, Banafsheh Saber Latibari, Anirudh Srikanth, Tyler Sheaves, Sayed Arash Beheshti-Shirazi, Sai Manoj PD, Satareh Rafatirad, Avesta Sasan, Houman Homayoun, and Soheil Salehi. 2023. Hardware Trojan Detection Using Machine Learning: A Tutorial. *ACM Trans. Embedd. Comput. Syst.* 22, 3, Article 46 (April 2023), 26 pages.
https://doi.org/10.1145/3579823

1 INTRODUCTION

Concerns regarding the security of manufactured ICs intended for use in sensitive applications have grown rapidly as a result of the use of untrusted parties throughout the IC supply chain network. The adversarial infestation of manufactured ICs with a Hardware Trojan (HT) is one of these security risks. A malicious alteration to a circuit intended to control, modify, disable, or observe its logic is known as a Trojan. There are more designers and design firms due to the expansion and globalization of IC design and development. IC design houses unable to manufacture their chips internally must use external foundries because building up a fabrication facility might easily cost more than $20 billion, and expenses for advanced nodes could be substantially higher. It can be difficult to build confidence with these external foundries, and they are often not trusted. The difficulties in the field of secure embedded system design were underlined by writers in the 2004 publication [1]. Some of the challenges described in [1] are: Processing Gap, Battery Gap, Flexibility, Tamper Resistance, Assurance Gap, and Cost, One of these security threats are the adversarial infestation of fabricated ICs with an HT. Conventional VLSI manufacturing tests and verification methods fail to detect HTs due to the different and un-modeled nature of these malicious modifications.

Due to the unique and un-modeled nature of these malicious modifications, conventional manufacturing VLSI test and verification procedures are ineffective in identifying HTs. This has motivated other researchers to look at methods for HTs detection using statistical analysis of side-channel data gathered from ICs, including side-channel power analysis [2], power supply transient signal analysis [3], regional supply currents analysis [4], temperature analysis [5], wireless transmission power analysis [6], and side-channel delay analysis [7–12].

The issue with many of the earlier HT detection techniques is the requirement for some golden model from which the parametric signature of the manufactured ICs can be gathered and used to define a decision boundary (power, delay, temperature, etc.) for separating the ICs infected with HTs. However, creating a golden IC is incredibly challenging, if not impossible: Most of the time, especially in advanced technology nodes, there is just one or a very small number of foundries available, and none can be trusted. Even if a reliable foundry were to exist, it would typically be too expensive to fabricate a small number of ICs in order to find a golden IC [13]. Another challenge is that a golden **integrated circuit (IC)** made in one foundry cannot be used to evaluate an IC made in another foundry. This is because each foundry has a very distinct manufacturing method.

For these reasons, we do not assume the existence of a golden IC or a golden model. Instead, we develop and train a learning-assisted timing-adjustment model combined with the STA acts as a golden model. This work illustrated in this tutorial is motivated by two previous papers: The side-channel power analysis in [13] and side-channel delay analysis in [7], a short description of which is given next section.

The side-channel statistical power analysis approach for HT detection in [13] proposed that the trusted region for the operation of a Trojan-free IC can be learned using a combination of

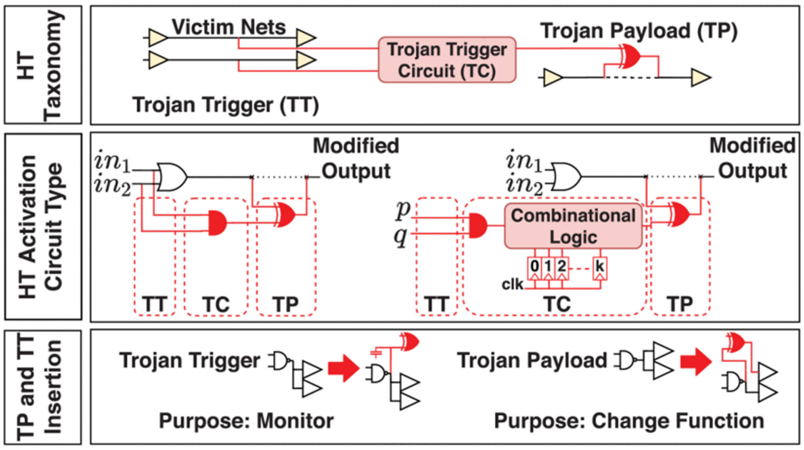

Fig. 1. HT taxonomy (top), HT activation circuit types (middle), and impact of inserted Trojan (bottom) [14].

measurements from the meticulously designed and distributed PCM structures, advanced statistical tail modeling techniques, and measurements from the trusted simulation model. However, this approach uses side-channel power analysis to find HTs. The size of the HT must be considered to see a significant change in leakage or dynamic power. As a result, this method cannot detect HTs constructed with a few gates. This is when the solution outlined in this tutorial can detect even a single added logic gate in a tested timing path. Additionally, to retrieve the process parameters, [13] uses PCMs (with a predetermined structure that is repeated and dispersed throughout the IC). The quantity and precision of PCMs are constrained, though. PCM falls short of precisely describing the behavior of various gates and metal layers, even if it can generally trace the process corner from chip to chip and be used for the crude calibration of timing and spice models. At this point, every timing path in our suggested solution could be used as a PCM to train the neural-assisted timing augmentation engine, allowing the effects of various timing path topologies, gate types and sizes, and variations in the capacitive or resistive load of various metal layers to be considered.

The side channel delay analysis solution in [7] utilizes the **Clock Frequency Sweeping Test (CFST)** to locate the HT. For a delay comparison, a Golden IC must be present, though. This work served as the basis for the development of the side-channel HT detection method (shown in this tutorial), which does not need a Golden IC but instead creates label data points for each feature set using CFST.

2 HT BENCHMARKS/TAXONOMY

Over the past decade, there has been a substantial increase in the study of HTs. Standard benchmarks for assessing HTs and their detection, however, were not yet available till the authors in [15, 16] developed a comprehensive HT benchmark set. In order to achieve this, they developed a resource suite with known HTs and "trust benchmarks" (benchmark circuits with HTs injected into them) that may be used by researchers in the community to examine different HT detection methods. In their study, they offer a thorough vulnerability analysis flow that can be used to produce these trust benchmarks in digital design at various levels of abstraction as shown in Figure 2. Additionally, they provide a thorough analysis of their benchmarks in relation to metrics like HT detectability and in the context of various attack methods. These HT suites are available at the Trust-Hub website [15].

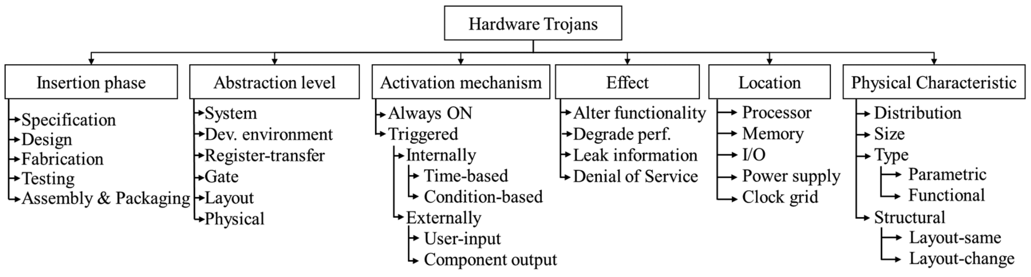

Fig. 2. HT taxonomy recreated from [15].

Some of the challenges, as mentioned in [16], that lead to this HT benchmarking effort were:

- Ad-Hoc Trojans: Most researchers, prior to a standardized benchmark suite, have resorted to using 'home-grown' HTs to showcase the advantages and accuracy of their proposed detection methods. Although these HTs may be well suited to a type of detection method, the outcomes when utilized in conjunction with other methods of detection could be very different. As a result, there was never a baseline for comparing the advantages of one detection technique to another when comparing the outcomes of different detection techniques. Furthermore, it is unclear if these HTs meet the fundamental requirement of an HT, namely that they must be able to completely avoid all conventional manufacturing test procedures including functional, structural, fault-based checks, and so on.
- Varying assumptions: The environment for simulation and implementation, as well as factors like the degree of process variation permitted, the difficulty of triggering HTs, HT switching activity, the size of the design (number of gates, HT size, etc.), and others, differ significantly between techniques. This makes it even harder to compare different HT detection methods.
- Ad-Hoc metrics: A lot of ad-hoc measures have also been employed to assess detection techniques. While some researchers may offer false positive/false negative rates and some may explain their findings in terms of test coverage, others may choose to evaluate their technique in terms of an arbitrary percentage detection rate. Even though various strategies might use the same HT attack model, comparing them using ad-hoc figures of merit is challenging.

2.1 HT Threat Model

In this section, we will briefly review the IC supply chain and discuss the HT threat model. Figure 3 shows the traditional IC supply chain model with the associated HT threats. Once the design specifications are decided by the chip architects, the design is sent to the appropriate design teams. Most design houses use **third-party IPs (3PIP)**, **third-party EDA tools or vendors (3P-EDA)**, and external design expertise through the IC design and implementation phase. As shown in Figure 3, there are a number of HT security vulnerabilities throughout the supply chain of an IC. The most common insertion point is at the untrusted fabrication facility where the adversary may modify the lithography masks to maliciously modify the IC. Some other threats are the insertion of HTs by 3PIPs, 3P-EDA vendors, and rogue designers. The threat model here assumes that the untrusted foundry is the adversary with access to GDSII, and HTs are potentially inserted into every IC that is produced.

The adversary's objective is to introduce an HT that is activated by a combination or a series of unusual occurrences. Figure 1 shows the components of an HT: (1) **Trojan's Trigger inputs**

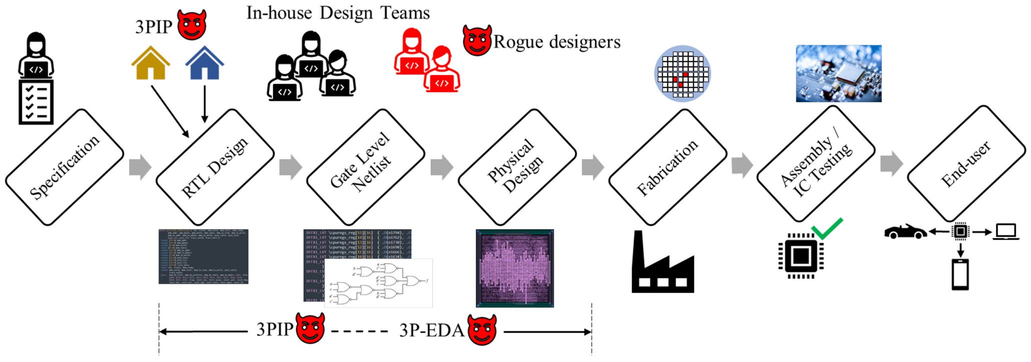

Fig. 3. Vulnerabilities of IC supply chain to HTs.

(TT), (2) **Trojan's Triggering Circuit (TTC)**, which may be sequential or combinational, and (3) **Trojan Payload (TP)**. The TP modifies the circuit's functioning after activation. Another assumption that is made, is that the design house has access to a secure test facility to test for HTs, and no Golden IC is available.

2.2 Current State of the Art in HT Detection

Researchers have investigated various methods of detecting these HTs through statistical analysis of side-channel information collected from ICs. Some of these side channels are:

- Side-channel Power analysis
- Power supply transient signal analysis
- Regional supply current analysis
- Temperature analysis
- Wireless transmission power analysis
- Side-channel delay analysis

Many of the existing methods require some golden model or Golden ICs that have been fabricated at a trusted facility and are free of HTs. A signature from these golden ICs is extracted and used to detect ICs with HTs. Using golden ICs as a reference has proved to be effective in augmenting the HT detection process, however, fabricating these Golden ICs is difficult as the design house has to have an in-house foundry or access to a trusted foundry.

3 HT DETECTION CHALLENGES

This section provides an overview of some significant challenges in detecting HTs. While the TP of the Trojans studied in this work adds at least one gate delay to their victim time route, the TT of the Trojans causes an extra capacitive load on the driving cell, slowing the rise and fall. Note that the added delay could be more than a single gate; this is because a large and complex TC may also affect the timing path hosting the TP if the sum of worse-case trigger sub-path delay and TC delay is larger than the delay of sub-path leading to the TP. In this paper (to address a more challenging scenario), we assume that TC is small, and the increase in the delay of the timing path hosting the TP is limited to a single gate delay.

Our scheme, ML-HTD (to be discussed in later sections), relies on side-channel delay analysis. It detects HTs by tracking and analyzing the changes in the delay resulting from tested timing paths. Our scheme does not rely on the availability of a Golden IC but on the timing model generated using **Static Timing Analysis (STA)** at the design time. However, the STA data can

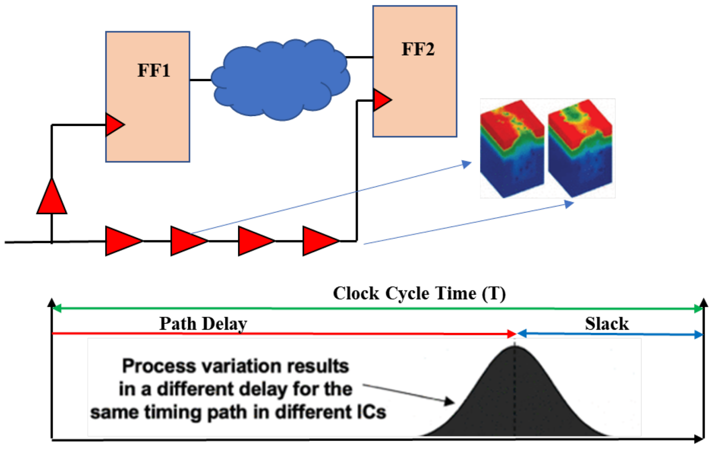
FF1
FF2
Clock Cycle Time (T)
Path Delay
Slack
Process variation results in a different delay for the same timing path in different ICs

Fig. 4. Figure shows the effect of random **Process Variations (PV)** on the delay of a timing path when tested across multiple dies.

differ significantly from the delay information calculated at the test time. The difference is due to pessimistic margins in generating GDSII files to account for various sources of variability, including Process Variation and process drift. What follows discusses the sources of such variations, and how they can be exploited by an adversary to insert stealthy HT. Then, in a later section, we describe the modeling of different sources of variability and how to improve the probability of HT detection by mitigating or modeling their impact.

3.1 Process Variation

The random process variation refers to the variations in the physical and electrical properties of transistors due to the physical limitations faced during the fabrication process. The random process variation impacts the delay and drive strength of fabricated transistors and makes HT detection more difficult as the test engineer needs to differentiate between the delays imposed by random process variation and the timing impact of an HT. Figure 4 illustrates the effect of the random process variation on the slack of timing paths.

3.2 Process Drift

The standard cell libraries used in a physical design house are characterized using the SPICE models for the manufacturing process at a new technology node, which is made public shortly after the process has reached a level of consistency. To ensure a high yield, the SPICE model and common cell libraries are padded with conservative margins. Additionally, the foundry updates the process by introducing newer and more capable stepping devices over time to increase yield and decrease cost. As a result, the manufacturing process and the published SPICE model diverge over time. A manufactured IC developed using the more outdated SPICE model has significant space thanks to process improvement. This method creates a security issue since an attacker in an unreliable foundry may utilize these wasted and concealed timing slacks to create stealthy HTs. The effect of Process Drift on the timing routes' slack is seen in Figure 5.

3.3 Voltage Noise

The **Power Delivery Network (PDN)**, which is an RLC network on an ASIC chip, responds to changes in the current demands of transistors by imposing voltage (IR) drop and voltage

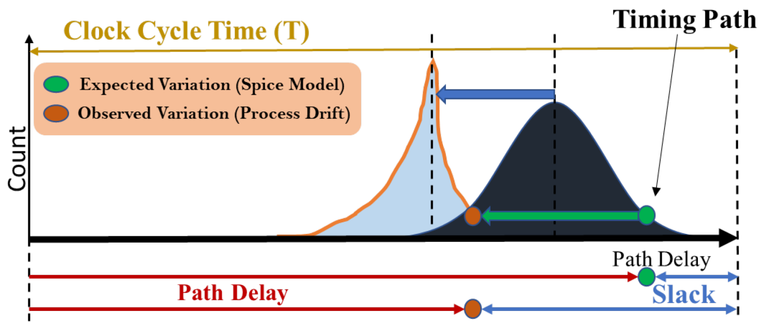
Clock Cycle Time (T)
Timing Path
Expected Variation (Spice Model)
Observed Variation (Process Drift)
Count
Path Delay
Path Delay
Slack

Fig. 5. Process improvement over time non-linearly alters the latency of various timing paths (process drift). Each timing path is distinctly impacted by process drift.

variation on transistors [19]. Modeling the IR drop and voltage noise during STA involves (1) setting a rail voltage value lower than the provided voltage to account for the IR drop, and (2) employing register-endpoint uncertainty to prevent voltage-variation-induced clock network [17]. To account for the worst-case scenario (and avoid setup/hold failure), pessimistic values for the rail voltage and uncertainty should be used. Voltage noise and IR drop are often less of an issue for timing paths. As a result, there is a security risk since the test engineer and physical designer are not aware of the significant amounts of timing slack that is built into the majority of timing paths due to pessimistic margins. A malicious party in a foundry that is not trusted may construct an HT and conceal the impact of the delay using the unused timing slacks.

4 ML-BASED HT DETECTION

In this section, we first answer why and how ML could help us in HT detection. Then we review the previous works that exploit different **Machine Learning (ML)** algorithms for Trojan detection.

4.1 Why ML Is Necessary For HT Detection?

HTs are difficult to detect by conventional testing and approaches because of the following reasons:

- Today's ICs are huge and complex.
- HTs are minute and difficult to detect.
- Finding the HT early in the design stage is important.
- Reducing HT detection cost.

Because of these, we can automate the process using different ML approaches like Regression, Deep Neural Networks, Graph Neural Networks, and Reinforcement Learning. HTs have specific features that can be given as input to an ML approach to do the detection. ML-based Trojan detection has two main phases: the learning phase and the detection phase [18]. Before we get into the details of a path-delay-based Trojan detection method, we give an overview of the current state of the art in ML.

4.2 Overview of ML Methods and Algorithms

ML is learning from experience. A reasonable amount of data is needed to use as input to our learning model. It has two main steps: Model training and Prediction. Figure 6 shows the current state-of-the-art ML approaches. We can categorize them into four main categories: Supervised, Semi-Supervised, Unsupervised, and Reinforcement. We briefly explain each group. Supervised ML uses input-output pairs to learn a model to map the input to output. So, an optimal

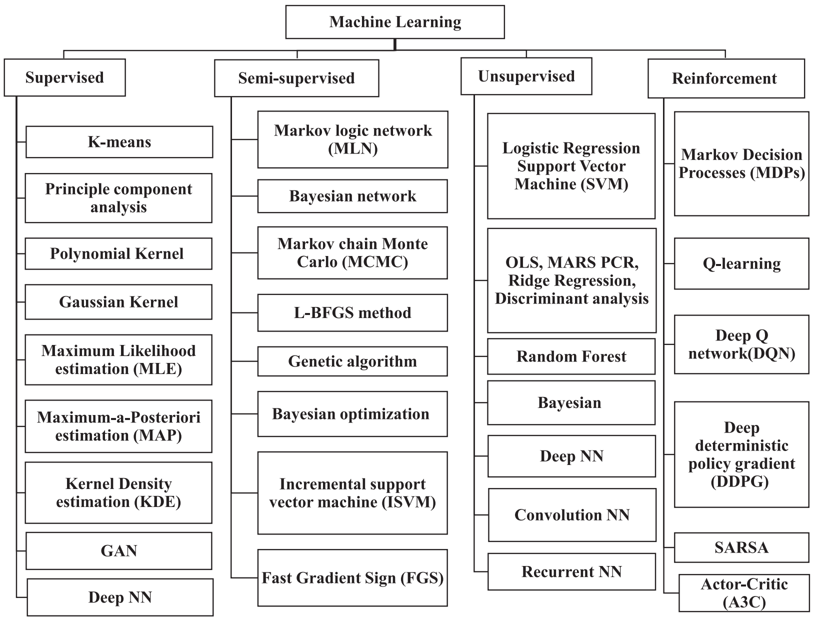
Machine Learning
Supervised
Semi-supervised
Unsupervised
Reinforcement
K-means
Principle component analysis
Polynomial Kernel
Gaussian Kernel
Maximum Likelihood estimation (MLE)
Maximum-a-Posteriori estimation (MAP)
Kernel Density estimation (KDE)
GAN
Deep NN
Markov logic network (MLN)
Bayesian network
Markov chain Monte Carlo (MCMC)
L-BFGS method
Genetic algorithm
Bayesian optimization
Incremental support vector machine (ISVM)
Fast Gradient Sign (FGS)
Logistic Regression Support Vector Machine (SVM)
OLS, MARS PCR, Ridge Regression, Discriminant analysis
Random Forest
Bayesian
Deep NN
Convolution NN
Recurrent NN
Markov Decision Processes (MDPs)
Q-learning
Deep Q network(DQN)
Deep deterministic policy gradient (DDPG)
SARSA
Actor-Critic (A3C)

Fig. 6. ML taxonomy (adapted from [19]).

trained model predicts output values that are the same as class labels. Semi-supervised ML uses a combination of a labeled and unlabeled dataset for model training. Unsupervised ML only uses untagged data to find a model to predict input data. The final group is Reinforcement learning which is based on using an agent to do some action in the problem search space based on the feedback it received from the environment to maximize the reward. K-means and GAN are famous models from a supervised group, the Bayesian network and Genetic algorithm is from a semi-supervised group, Random Forest, Regression, and deep learning approaches are unsupervised based ML and Q-learning, SARSA, and Actor-critic are well-known RL-based algorithms.

4.3 A Review on Previous ML-based HT Detection Methods

Figure 7 shows the overview of ML-based HT detection. We can categorize the ML-based HT detection approaches into the following groups:

4.3.1 Classification Approaches Using Netlist. These approaches classify design nets into Trojan and Not-Trojan. Authors in [18] proposed an HT classification method to identify the HTs at the gate level. They used a **support vector machine (SVM)** and a **neural network (NN)** as their ML model. Using the proposed model, they detect the HT Nets. They extracted five features and considered them as a five-dimensional vector. In [20], they introduced 51 HT features and used a Random Forest to extract the 11 important features to improve their ML-based HT classifier.

4.3.2 Reverse Engineering Based Approaches. Reverse engineering consists of five steps. The first three steps are (1) Decapsulation, (2) Delayering, and (3) Imaging for layout identification. And the last two stages extract the netlist. In [21], they used One-class SVM to propose a reverse engineering-based HT detection. This work extracted the features from the IC images directly hence avoiding extracting netlists. By skipping the netlist generation steps, they skip unnecessary efforts. In [22], they presented a technique to automate the layout identification. In this work,

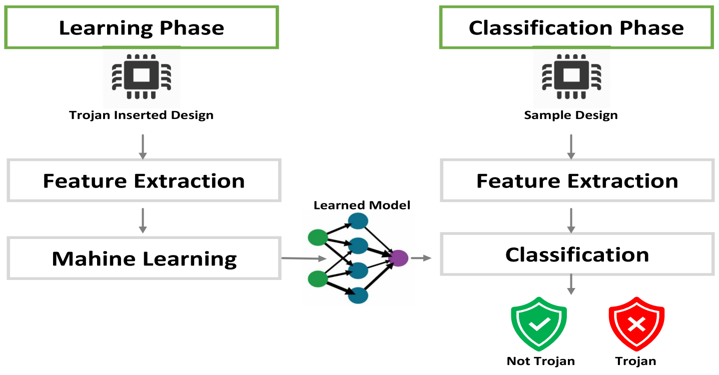

Fig. 7. Overview of ML-based HT detection.

they used a histogram of oriented gradients to extract features of the circuit layout. The extracted features are fed to the decision tree classifier. Moreover, they leveraged AdaBoost to benefit from a stronger classifier named **Automatic HT Detection and description Tool (AHTDT)**. In [23], they considered HT detection as a clustering problem, and they used k-mean clustering as a solution. Compared to the SVM-based method, this approach is not dependent on the choice of parameters, and as a result, it is easier to train. To implement their method, they divided the image of the layout into grids, and then they extracted features from each grid to form feature vectors. Finally, they applied the K-means clustering approach to group these feature vectors into k clusters.

4.3.3 Golden Model Free Approaches. Using the golden model is expensive, so, the approaches in this category eliminate the need for a golden model for Trojan detection. In [24] the author used **Controllability and Observability** to propose **COTD**. The COTD uses k-means clustering to cluster the HT signals into Trojan-inserted and Trojan-free. In [25], they formulated the HT detection algorithm as a two-class classification problem, and they used transient power of the simulated ICs during IC design flow for training the model, so the proposed model does not need HT real-time behavior. They tried different regression learning algorithms and also used adaptive iterative optimization of one algorithm, matched algorithm pairs, and cost-sensitive detection and suitable algorithm settings against process variations for optimization.

4.3.4 LASCA. In [26], the authors present a **Learning Assisted Side Channel delay Analysis (LASCA)**. The proposed neural network works as a process tracking watchdog to correlate the extracted static timing data at the design time to the extracted delay information from clock frequency sweeping at test time to detect the HT. The main idea of this work is based on the fact that the **Trojan Trigger (TT)** part of an HT makes the rise and fall of the design slower, and the **Trojan payload (TP)** of it adds a gate delay to the timing path that it is inserted to. LASCA exploits several techniques to mitigate the impact of voltage noise, **Process Variation (PV)**, and process drift to improve the correlation between the timing model and IC behavior. They modeled the timing impact of process drift using a process tracking NN-watchdog. The proposed NN module predicts the difference between the reported slack by STA at design time and the extracted from IC at test time. To model and mitigate PV, at first, they divided the PV into two categories (1) Random Class (independent intra-die PV) and (2) Persistent class (inter-die and correlated intra-die variation). For the first class, they used speed binning on fabricated ICs with the assumption that ICs in the same bin are similarly affected by the persistent PV. For the second class to reduce randomness, they averaged the delay of timing paths collected from many IC in the test set. To model voltage noise, they followed the IR-ATA methodology. AVATAR [14] is an extension of the side-channel delay-based HT detection method, LASCA, and will also be discussed wherever appropriate in this tutorial paper.

4.3.5 Deep Learning Based Approaches. In HERO [27], the authors provide a holistic solution for HT detection using a DNN model and by exploiting many different types of features that have been generated in different manufacturing stages (design, verification, post-silicon, etc.) from a variety of benchmarks. Hero has a pipeline to combine and handle the different sources of data and converts them into a uniform format (single-channel images), and finally apply the DNN. So, converting features to the image provides the possibility of using various pre-trained deep convolution-based networks. Hero also uses data augmentation to fix the dataset distribution imbalance. Hero uses several HT techniques, including Gate level netlist analysis, **Placement and Routing (PNR)**, and Side channel analysis. In [28, 29], they used the graph data structure to represent the hardware design and generate the **Data Flow Graph (DFG)** for both RTL codes and gate-level netlists and **Graph Neural Network (GNN)** to detect Trojans. The main goal of this paper is to find a feed-forward function f to detect whether an HT is inserted into the design. The presented approach is based on three main steps: (1) extracting DFG from hardware, (2) Passing DFG to GNN for graph learning and feature extraction, and (3) Classification using **Multi-Layer Perceptron (MLP)**. Authors in [30] used **Generative Adversarial Neural Network (GAN)** in combination with **Stacked Auto Encoder (SAE)** for HT detection. To elaborate, they used GAN for data pre-processing in order to generate fake samples identical to the original Trojan-inserted samples to keep the dataset balance. Next, they used SAE as a classification model.

4.3.6 Using Reinforcement Learning (RL) for HT Detection. In [31], the authors propose AdaTest, an adaptive test pattern generation framework for HT detection that is saleable and has noise and variation resistance. AdaTest used RL to generate test inputs and adaptive sampling to prioritize test samples that provide more information for HT detection. AdaTest framework has two main phases: (1) Circuit Profiling and (2) Adaptive Test pattern Generation. During the first phase, AdaTest characterizes each node in the design based on transition probability and SCOAP testability. In the second phase, it used a reward function that gives the reward according to the number of times each rare node is triggered and the SCOAP testability measure of the rare nodes and graph-level distance of the circuit. AdaTest also used a hardware acceleration approach that pipelines the computation in online TPG and deployed circuit emulation to accelerate reward evaluation.

5 ML-BASED HT DETECTION IMPLEMENTATION

In our detection model, we assume that the adversary is an untrusted foundry with access to the **Graphic Database System format (GDSII)** of the design, aiming at inserting an HT that is triggered based on a combination or sequence of rare events. We assume that the HT has several Triggers and at least one payload. However, the HT detection solution implemented applies to HTs with no Payload (inserted for monitoring purposes). We further assume that the same HT is inserted in all fabricated dies. We also assume that the foundry can skew the process (making faster transistors) to create available slack for the insertion of HT in desired timing paths without making the overall delay of the timing path larger than the delay reported (or expected) by STA at design time. HT detection, in our solution, is performed in a trusted facility.

5.1 Clock Frequency Sweeping Test (CFST)

Frequency sweep tests fall squarely within the scope of analog electronics. The objective is to examine the effects of various circuit elements on a monochromatic input signal. The output voltage or current is then measured at each input frequency while the input signal's frequency is swept across the desired range. The phase shift applied to the signal by reactive circuits is also recorded along with the output voltage or current. Researchers in [32] first showed path delay analysis for uniquely identifying ICs. Authors in [7] have used clock frequency sweeping tests to detect HTs.

5.2 HT Detection Flow

Figure 8 gives the overall flow of the **ML-Assisted HT Detection (ML-HTD)**. The design stage is augmented with an additional stage for statistical modeling for IR drop and voltage noise using IR-ATA flow as described in [33]. As a result, based on its estimation of voltage drop and voltage noise, the STA reports the timing slack of each timing path (as opposed to a global pessimistic margin). This will enhance the correlation between the timing slack detected during testing using CFST and the timing slack anticipated by the timing engine, as we shall demonstrate in the results section. The foundry is then contacted to fabricate the final GDSII. The unreliable foundry may test the functionality of the manufactured ICs. After that, the functional ICs are delivered to a reliable facility for Trojan detection.

We need to locate the TT/TP-induced slack shift in order to detect a Trojan. As seen in Figure 1, a TT increases the capacitive load on the observed net's driving cell, while a TP adds an extra gate delay to each timing line that goes through its victim net. We must include all nets in our delay study because we have no prior information on which nets are affected and need to identify a victimized or monitored net (by a TP or TT). A P2P wire is a net that joins a driver cell's output pin (or principal input) to one of its fanout cells' input pins (or primary output). So a gate with a fanout of four has four P2P wires. Each P2P wire will be tested for rise and fall transitions.

This procedure may be performed for N separate timing paths going through that net to boost the detection rate and consider PV. The maximum frequency of the tester equipment serves as the second criterion for timing-path selection. The delay of the chosen paths should be greater than the restriction imposed by the tester equipment's maximum achievable frequency. The P2P wire is considered a potential for Trojan detection by power-based detection techniques if it is not in a timing path that is long enough for CFST. The power-based HT detection algorithms (such as [34–37]) that depend on full or partial activation of such paths are appropriate for timing paths with a small number of gates (in their data sub-path), as they have high controllability. We create the **Path Delay Fault (PDF)** test vectors using an **automatic test pattern generation (ATPG)** technique for all other timing-path possibilities. The path is chosen differently if ATPG cannot produce a test pattern for a particular path. Any path through the P2P wire that ATPG cannot generate a test vector for is eliminated.

5.3 HT Detection Methodology

5.3.1 Modeling and Tracking the Process Drift. Different timing-paths experience a non-uniform shift in delay as a result of process drift. We train an NN that serves as a process tracking watchdog (NN-watchdog) to model the timing impact of process drift. The gap between the delay indicated by STA at design time and the measured delay from the manufactured IC at test time is predicted by this NN-watchdog. We employ a labeled data set with each labeled data point consisting of a collection of 48 input feature values and a label (output) value to train the NN-watchdog. The input characteristics, detailed in Table 2, are taken from EDA tools for timing engines and physical design.

We modeled the process drift (and systematic process variation) by extracting the shift in delay values from SPICE simulations, done using a skewed SPICE model, to evaluate the efficacy of NN-watchdog (and for the lack of access to produced ICs). To do this, we first retrieved the SPICE model from the input training for each timing path. The SPICE model was then skewed so that the NMOS and PMOS transistors were X% quicker and the Metal capacitance for Metal layers 1 to 7 derated by Y% in order to replicate a systematic process drift. We get a consistently quicker

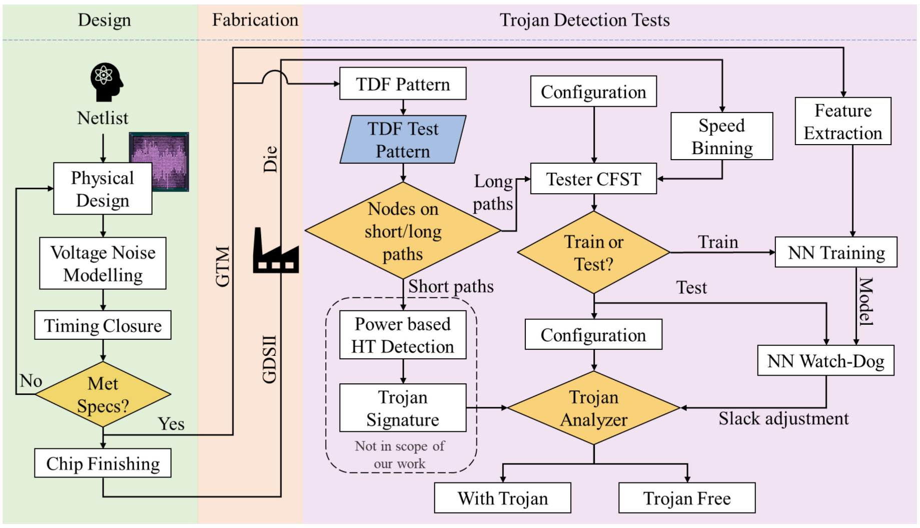

Fig. 8. ML-assisted HT detection (ML-HTD) flow.

```
1:  $NP \leftarrow mR^2$        ▷ R is the registers count, and $m$ is a large number (e.g. 10)
2:  $TimingPaths \leftarrow$ Select $NP$ timing-paths (min of $m$ path per endpoint)
3:  for all $path$ in $TimingPaths$ do
4:      feature($path$) ← Extract $path$ features from GTM        ▷ input feature
5:      GTM($path$) ← Extract $path$ slack from GTM
6:      $Slack$(path) = 0
7:  for all $die$ in $Dies$ do
8:      for all $path$ in $TimingPaths$ do
9:          CFST($die$,$path$) ← Slack of $path$ in CFST test of $die$
10:         Slack($path$) += CFST($die$,$path$)
11: for all $path$ in $TimingPaths$ do
12:     Slack($path$) = Slack($path$)/$NP$;
13:     $\Delta_{slack}(path)$ = Slack($path$) - GTM($path$)        ▷ label
14:     data-points($path$) ← (features($path$),$\Delta_{slack}(path)$)
```

Fig. 9. Generating a training set for the NN-Watchdog.

or slower process model depending on how X and Y are chosen. For instance, in our simulations, using (X, Y)=(5,5),(0,0),(5,5) results in Fast, Typical, and Slow process models, respectively.

In this paper, we have evaluated three different models for predicting the process-induced change in the timing path delays. Details of each model are discussed in the relevant section:

(1) Linear Regression (Ridge Regression) Model (Baseline) Ridge Regression [38] is a regularized linear regression model and it is useful for modeling and tracking multicollinearity phenomena.
(2) Multi-Layer Perceptron Regression Multi-Layer Perceptron (MLP), is a non-linear neural network composed of an input layer, one or more hidden layers, and an output layer. Details and setup of the MLP regressor used in this paper are summarized in the table.
(3) Stacking Regression Model The structure of the Stacking Regression model [38], which is also known as stacked generalization [38], is depicted in Figure 14. Stacking Regression is an ensemble learning technique in which different estimators are arranged into two layers to form a regressor with lower variance in comparison to each (member) regressor.

```
1: $N$ = # paths to be tested through each net in the design
2: $Nets \leftarrow$ all nets in the design.
3: for all $net$ in $Nets$ do                ▷ net selection of Path Delay Fault (PDF) test
4:     $TimingPaths +=$ select $N$ timing-paths passing through $net$
5: Perform speed binning on all dies and assign them to B bins.
6: for all $bin$ in $B$ do                                                   ▷ NN training
7:     $NN_{bin} \leftarrow$ Train a NN-Watchdog according to the algorithm 1
8:     $\sigma_{NN_{bin}} \leftarrow$ the standard deviation of $NN_{bin}$
9:     for all $die$ in $bin$ do
10:        Slack = 0
11:        for all $path$ in $TimingPaths$ do
12:            CFST(bin,die,path) $\leftarrow$ $path$ slack measured by CFST $die$ in the $bin$
13:            Slack(bin,path) += CFST(bin,die,path)
14:    for all $path$ in $TimingPaths$ do
15:        $\mu_S$(bin,path) = Slack(bin,path)/sizeof(bin);
16:    $T_{Th}$= $4 \times \sigma_{NN_{bin}}$        ▷ Detection Threshold $= 4\sigma$ to reduce false positive
17:    for all $path$ in $TimingPaths$ do
18:        GTM(path) $\leftarrow$ query the slack of path from GTM
19:        NNSD(path) $\leftarrow$ slack shift suggested by $NN_{bin}(path)$
20:        AS(path) = GTM(path) + NN$_{Watchdog}$(path)                        ▷ Adjusted Slack
21:        $\delta$ = $\mu_S$(bin,path) - AS(path)                ▷ Shifted delay after adjustment
22:        if $(\delta > T_{Th})$ then                                        ▷ Trojan Classifier
23:            Likely Trojan Set $\leftarrow$ $path$
```

Fig. 10. ML-assisted HT detection (ML-HTD) algorithm.

Table 1. Description of Models used and Model Hyper-parameters

Model	Hyper-parameters
MLP	in_layer=48, hidden_layer=23, out_layer=1, activation='tanh', optimizer='Adam', learning_rate='adaptive', start_lr='0.1'
Random Forest	n_estimators='1024', bootstrap='true', min_leaf='1', min_split='2'
XgB	n_estimators='1024', learning_rate='0.05',
Lasso	alpha='1', max_iter='5000'
Ridge	alpha='1', max_iter='500'
ElasticNet	alpha='0.001', max_iter='1000'

More precisely, at a two-layer stacked regressor, we used regressors XGB [39, 40], Enet [41–43], Lasso [44, 45], Ridge [46], MLP [38, 47], and RandomForest [48, 49] for our first layer regression. The predictions of these regressors, yˆ1 to yˆ6, are stacked together and fed to the second layer of regressor(s). In general, the second layer may also consist of multiple regressors. The overall prediction $\hat{y}$fin is obtained by averaging the results of the second layer regressors. In our model, we have only deployed a single Lasso [38] regressor in the second layer. Including additional regressor results in only negligible improvement in the model's prediction performance at the cost of increased complexity. A detailed explanation of the fundamentals of the all models used will be given in the regression models section.

5.4 Feature Selection, Extraction, and Dataset Generation

To detect an HT, we need to detect the change of slack in affected timing paths due to TT or TP presence. As Figure 1 shows, the TT adds capacitive load to the driving cell of an observed net, and the TP inserts one (or more) additional gate(s) in the victim net. Without a Golden IC, we do not know which nets have been victimized. Hence, we need to check for the delay change of timing paths by investigating each net included in the suspicious timing paths, i.e., the timing paths whose slack appeared to be larger than expected when doing the frequency sweeping test. We define a **Pin-to-Pin Wire (P2P-wire)** as a net connecting the output pin of a driver cell (or a primary input) to the input pin of one of its fanout cells (or a primary output). Hence, a gate with a fanout of 4 has 4 P2P wires. Each P2P wire will be tested for rise and fall transitions. To increase the detection rate and to account for random process variation, this process may repeat for N different timing paths passing through each net (a similar approach to N-detect testing).

Table 2. Description for Each of 48 Features, Extracted From Each Timing-Path to Build the NN Training Set

Total of 48 features, three features extracted from each timing-path		
Setup Time	Path Delay reported in STA	Sum of fanout over cells in DP
Total of 45 features extracted, 15 features from sub-path (CP, LP and DP)		
number of gates	subpath delay	# cells of x0 strength
# cells of x1 strength	# cells of x2 strength	# cells of x4 strength
# cells of x8 strength	# cells of x16 strength	# cells of x32 strength
Total of Length of M1	Total Length of M2	Total length of M3
Total Length of M4	Total Length of M5	Total Length of M6

(LP: Launch Portion of Timing-Path, CP: Capture Portion of Timing-Path, DP: Data Portion of Timing-Path, M: Metal Layer, x: Drive Strength of the Gate).

The second criterion for selecting the timing paths is the maximum frequency of the tester equipment; the selected paths' delay should be larger than the limit imposed by the maximum reachable frequency of the tester equipment. If the P2P wire in no timing path is long enough for CFST, it cannot be subjected to side-channel delay testing. However, such timing paths could be used as a candidate for HT detection via power-based detection schemes. This is because timing paths with a smaller number of cells often exhibit better controllability and are better suited for the power-based HT detection schemes (e.g., [50, 51]) that rely on full or partial activation of an HT. We generate the **Path Delay Fault (PDF)** test vectors for the long timing-path candidates using an **Automatic Test Pattern Generation tool (ATPG)**. Suppose the ATPG fails to generate a test pattern for exclusive PDF testing of a given net in a suspicious timing path (in other words, no other timing path exists that contains the targeted net while it does not contain any other nets in the suspicious timing path). In that case, a sequence of nets (preceding or proceeding the target net) is selected for test pattern generation.

5.5 Regression Models

5.5.1 MLPRegressor. A Multi-Layered Perceptron, as shown in Figure 11, is a simple neural network with three main components, the input layer, the hidden layer, and the output layer. The input is fed to the input layer through the neurons. Each and every neuron is connected through weights which represents the strength of the connection between two neurons. The neurons in the layers use an activation function to decide whether the value carries useful information to pass on to the next layer which will help with the prediction. The neural network learns iteratively by backpropagating the overall error, as shown in Equation (1), i.e., the difference between the predicted value and the ground truth value. This step is performed by an Optimizer function which helps in effectively finding the best fit for the model on the training set. After the network gets trained over a set number of epochs, the model is tested by feeding the inputs from the testing set to the input layer and is passed through successive layers, and ultimately we get the desired output.

$$Loss\ (MLP\ Regression) = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 \tag{1}$$

$$where,\ M\, is\, the\, number\, of\, instances\, and\, p\, is\, the\, number\, of\, features$$

5.5.2 Ridge. Ridge regression is a way to limit the number of independent variables (columns/features/attributes) in the regression. The regular least-squares criterion minimizes the

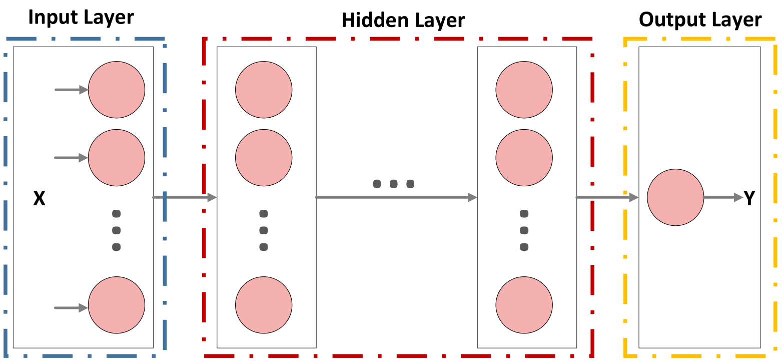

Fig. 11. Multi-layer perceptron regression model.

least-squares of the error plus a regularization term that is a product of a constant and the sum of squared coefficients, as shown in Equation (2). The main purpose of a Ridge regression model is to penalize the size of the weights to avoid running into over-fitting the model. It also minimizes the effect of irrelevant features to make the predictions more accurate. The Ridge regression is most effective when dealing with a dataset that has multicollinear features. We train our models with 45 interdependent features. Hence, using the Ridge regression helped us reduce the prediction variance.

$$Loss\ (Ridge\ Regression) = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda_1 \sum_{j=0}^{p} w_j^2 \tag{2}$$

5.5.3 Lasso. Lasso regression does pretty much the same thing as Ridge regression, it is another way to limit the number of independent variables in the regression. Lasso regression performs L1 regularization, i.e., a penalized term will be added whose value is equal to the absolute value of the magnitude of the coefficient, as shown in Equation (4). This algorithm is best used when the low-contributing features can be eliminated. Using the L1 penalty avoids over-fitting by selecting useful features to predict the output class.

$$Loss\ (Lasso\ Regression) = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \lambda_2 \sum_{j=0}^{p} |w_j| \tag{3}$$

5.5.4 ElasticNet. The ElasticNet regression combines the behavior of the Ridge and Lasso regression models. The loss function includes the effect of both L1 and L2 regularization terms. Since the Ridge regression model does not guarantee to suppress the contribution of all the irrelevant features, the effect of Lasso regression is combined with the Ridge regression term to produce robust results. The α hyperparameter is used to control the weighted combination of the L1 and L2 terms. The ElasticNet model outperforms both Lasso and Ridge regression models while handling the bias and complexity of the model.

$$Loss\ (ElasticNet\ Regression) = \frac{1}{M}\sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{p} w_j * x_{ij}\right)^2 + \alpha * Lratio \sum_{j=0}^{p} w_j^2 + 0.5\alpha * (1 - Lratio) \sum_{j=0}^{p} |w_j| \tag{4}$$

$$where,\ \alpha = a + b\ and\ Lratio = \frac{a}{a+b}$$

5.5.5 Random Forest. It is a type of supervised ensemble learning model that uses a combination of different decision trees and the final outcome is determined by averaging the results of all each and every Decision Tree. The Decision Trees start with the root of the tree and follow

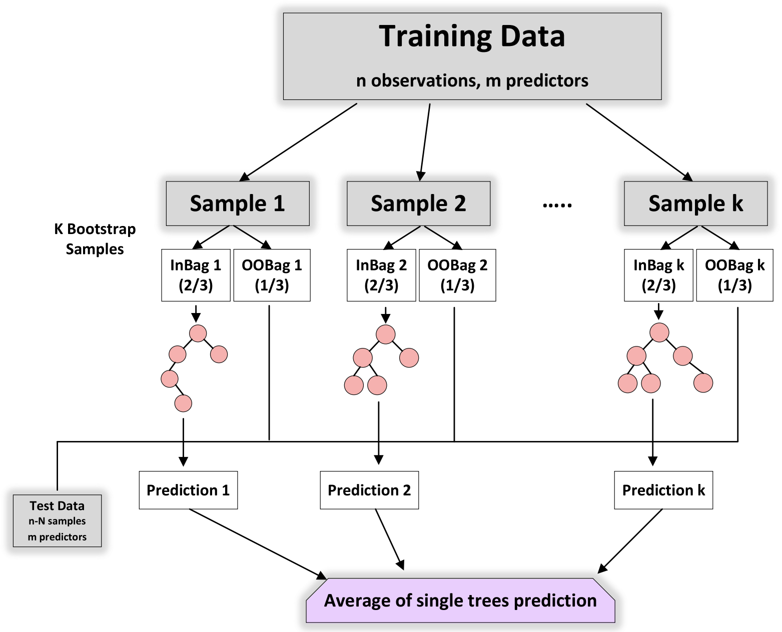

Fig. 12. Random forest regression model.

splits based on variable outcomes until a leaf node is reached and the result is given. It is assumed that each and every Decision Tree learn different features and are almost independent of each other.

As the size of the dataset increases, the Random Forest models are prone to over-fitting. This can be avoided by incorporating the bootstrapping technique along with the ensemble learning algorithm to produce desirable results. Bootstrapping is the process of randomly sampling subsets of a dataset over a given number of iterations and a given number of variables. These results are then averaged together to obtain a more powerful result, as shown in Figure 12.

5.5.6 Gradient Boosting. Gradient boosting is one of the variants of ensemble methods where multiple weak models are created and combined to get better performance as a whole. It is powerful enough to find any nonlinear relationship between the model target and features and has great usability that can deal with missing values, outliers, and high cardinality categorical values on the features without any special treatment.

The goal is to teach a model to predict values by minimizing the mean squared error. At any stage, the model tries to improve the imperfect model to give a better output. A residual function is calculated to be added as an estimator to the current output. This is used to improve the model over several iterations. The next model attempts to correct the errors of its preceding model by using a Mean-Squared Error loss function. The gradient of this loss function is calculated with respect to the current model output. This is further used to fit the next weak-learner model. This is performed until a threshold is reached, usually the number of estimators, as illustrated in Figure 13.

5.5.7 StackingCV. Stacking is an ensemble learning technique that combines multiple regression models via a meta-regressor. Meta-regression is similar to general regressions, where the outcome is predicted through the values of different explanatory variables. In meta-regression, the outcome is the effect estimate of the first-level regressor model outcomes. The StackingCV

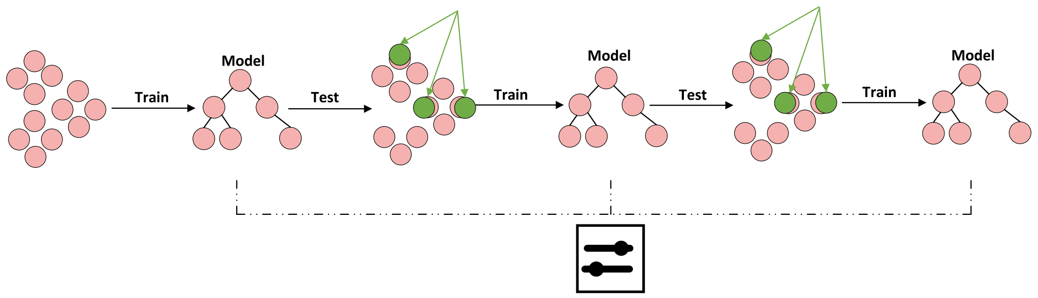

Fig. 13. Gradient boost regressor model.

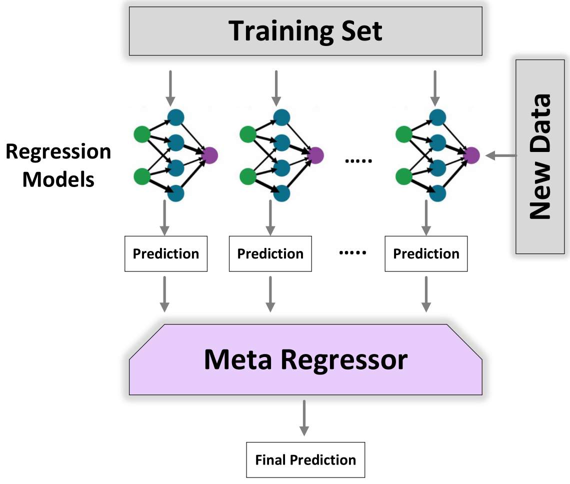

Fig. 14. StackingCV regressor model.

regressor extends the standard stacking algorithm (implemented as Stacking Regressor) using out-of-fold predictions to prepare the input data for the level-2 regressor.

As shown in Figure 14, the first-level regressors are fit to the same training set that is used to prepare the inputs for the second-level regressor, which may lead to over-fitting. The StackingCV Regressor uses the concept of out-of-fold predictions: the dataset is split into k folds, and in k successive rounds, k-1 folds are used to fit the first level regressor. In each round, the first-level regressors are then applied to the remaining one subset that was not used for model fitting in each iteration. The resulting predictions are then stacked and provided as input data to the second-level regressor. After the training of the StackingCV Regressor, the first-level regressors are fit to the entire dataset for optimal predictions.

5.6 Model Training

In this section, we discuss the training setup, the regression model parameters, and hyper-parameters. All information is in a tabular form in Table 1. We leverage the Scikit-learn [38] library in Python to implement the regression models to detect the inserted HTs. The MLP model with 23 hidden layers and Adam optimizer [52] and an adaptive learning rate was found to produce good results. Our model was trained for 10,000 epochs since the training set was considerably large with a Tanh activation function. The dataset was trained on an 80:20 train-test split. Our model was able to perform well with an accuracy of on the test set. To implement our Ridge Regression model, the 'α' hyperparameter was set to 1.0 since that was found to be most optimal to fit on the dataset. We train our models with 45 interdependent features. Hence, using the Ridge regression helped us reduce the prediction variance.

To implement our Lasso Regression model, the 'α' hyperparameter was set at 0.001, and the model was trained for 5,000 epochs and was seen to yield the best results. For our Random Forest Regression model, the maximum number of trees used is 1,024. The Gradient Boost Regressor model was trained with a learning rate of 0.01 and a maximum number of 1,024 trees. We used the MLXtend library [53] in Python to implement our StackingCV regressor model. The Lasso regression model was used as a meta-regressor model. We used the Scikit-learn library in Python to implement our ElasticNet Regression model. We chose the 'α' term to be 0.001 and trained for 1,000 epochs, which is the default value.

5.7 Implementation Results and Discussion

In this section, we first describe our experimental setup, and then describe the results of our simulations:

Setup: We assessed the effectiveness of **ML-Assisted HT Detection (ML-HTD)** on the three major IWLS benchmarks [54] (Ethernet, S38417, and AES128). We created each benchmark and added 90 HT. These HTs are straightforward combinational HTs, consisting of a single 2-input XOR gate to transfer the Trojan payload to a target net, a single AND-tree to create the HT activation signal, and four input triggers attached to specific nets (as Trigger Nets). Nets are carefully picked from non-critical timing patterns with enough available timing slack to insert the Trojan trigger and payload. The positioning of the HT Circuit (and the first gate of the AND-tree) in relation to the triggering net determines the influence of the TT capacitive delay. The HT Circuit (first gate) is positioned within a 20 μm radius of the Trojan Trigger nets to minimize the Trigger impact (assuring a small delay impact). In order to reduce the influence of the driver gate's gated capacitance on the latency of the Trojan triggering net, the AND tree for the HT Circuit is likewise built using the smallest AND gate available in the standard cell library. We would have 90 placed-and-routed netlists for each benchmark, each including a single HT circuit, in order to demonstrate the sensitivity of our approach. Each HT Circuit is added in a distinct placed-and-routed netlist. Each benchmark has a physical design that is hardened and timing that is closed at 1.4GHz in 32nm technology.

We do not know if a timing path chosen for training contains an HT during NN-watchdog training. As a result, we also assessed the effect of adding Trojan-affected temporal pathways to the practice set. With the inclusion of 0, 1, 5, 10, and 15 Trojan pathways in their training set, we trained 5 NN-watchdogs. Our objective is to assess whether Trojan-affected temporal routes, such as trigger nets or payload nets, might contaminate the model to the point where HT evasion occurs when evaluated using ML-Assisted HT Detection (ML-HTD).

By adjusting the slack recorded for each timing path to the neighboring higher clock sweeping frequency step, and modeling the CFST step size, the silicon CFST test was simulated using SPICE simulation. Modern testing equipment allows for step sizes as tiny as 10-15ps. Therefore, we decided on 15ps for the tester's step size. Here, we additionally consider the influence of random process fluctuation (see Figure 4). In order to model the variation in path delays from chip to chip, each SPICE simulation is subjected to 200 Monte Carlo simulations (modeling CFST performed on 200 different dies in the same speed bin). For these simulations, the threshold voltage (Vth), oxide thickness (Tox), and channel length (L) are varied. In keeping with [7], we have limited the random process variation to 5%.

Table 3. Percentage of True Positives (T+) and False Positives (F+) when ML-HTD (as Described in 9) with NGTM-10 is used for Detection of T+ and F+ in Slow, Typical, and Fast Speed Bins

Benchmarks	Slow Bin		Typical Bin		Fast Bin		No-Binning	
	T+	F+	T+	F+	T+	F+	T+	F+
AES128	88.6	0.11	87.8	0.17	86.1	0.18	0.78	0.31
Ethernet	87.3	0.17	85.5	0.12	88.6	0.15	0.80	0.48
S38417	83.7	0.19	82.2	0.23	80.3	0.39	0.77	0.45

Table 4. Threshold Values used for Trojan Trigger and Trojan Payload

Benchmarks	TP		TT	
	Youden	4 x σ_{NN}	Youden	4 x σ_{NN}
AES128	27.1	29.86	16.3	29.86
Ethernet	35.5	38.67	15.4	38.67
S38417	24.7	27.46	17.2	27.46

Trojan detection in Fast-bin described in in Figure 10.

In our simulations, we assessed the effectiveness of HT detection using two mechanisms for building our reference (Golden) timing model:

5.7.1 Shifted STA (SSTA). In this case, in order to detect HT, we utilize the STA findings as our reference Timing Model. The process drift makes it very ineffective to use STA findings directly. We calculated a static shift value, acquired by averaging the observed shift from several sampled timing paths, and have moved all reported slacks by STA using this value to account for process drift in SSTA. The detection threshold for this method has been placed at 45ps, corresponding to the latency of a 2-input NAND gate in our standard cell library.

5.7.2 Neural Shifted Golden Timing Model (NGTM). Using the suggested NN-watchdog, the process drift and systematic process variation is modelled, see Table 3. The anticipated shift by NN-watchdog, which generates path-specific changes in slack based on path topology/features, is then added to the STA findings. We also assessed the use of MLP and stacked regression as NN-watchdogs to demonstrate the efficacy of the layered learning model. There is no assurance that the timing path(s) impacted by the HT will not be included in the dataset used to train the NN-watchdog. As a result, we have looked at how well NGTM performs when the training set comprises 0, 1, 5, 10, and 15 timing paths that are HT-affected. In this method, we have adjusted the threshold for HT identification to 4σ regressor standard deviation. The frequency of false positives is greatly decreased by selecting σ. A critical understanding of why the NN-watchdog created using the stacked-regression model is anticipated to be more sensitive/accurate than the MLP-regression model may be gained by comparing the standard deviation of the two models, as shown in Table 4. While statistically benefiting from a comparable false-positive rate, it benefits from a lower detection threshold.

We have extracted and reported the ideal threshold from the ROC curve using Youden's [55] method to assess the validity of the threshold values used for the HT detection flow outlined in this tutorial. Note that the ideal threshold can only be used for quality assessment since it depends on the ground-truth table (knowing precisely which time paths are and are not affected by HT). For both TT and TP-based ROC curves, the Youden technique gives a distinct detection threshold. The **Receiver Operating Characteristic (ROC)** curve is a plot between the False

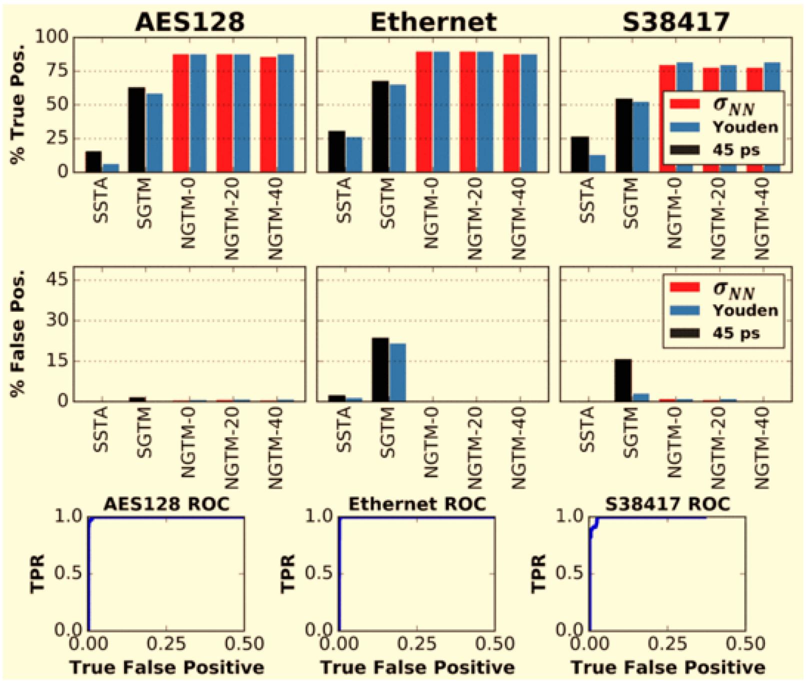

Fig. 15. Trojan Payload detection results.

Positive Rate and the True Positive Rate of a classification model. These plots are often used as a part of diagnostic test studies to demonstrate the trade-offs between Specificity (True Positive Rate) and Sensitivity (False Negative Rate) and hence find the cutoff point which optimizes Specificity and Sensitivity. The True Positive Rates and the False Negative Rates of a model are calculated for different cutoff points. Youden's index is one of the sought-after metrics to find the optimal cutoff from the ROC curve. Youden's index combines sensitivity and specificity into a single measure ($J = Sensitivity + Specificity - 1$) and has a value between 0 and 1. In a perfect test, Youden's index equals 1. It is comparable to the vertical distance from the ROC curve for a single choice threshold to the diagonal no discrimination (chance) line. The overall accuracy of the diagnostic test is determined by the **Area Under the Curve (AUC)** metric. The ROC curve comes in handy when comparing the results of different models. An excellent model has an AUC near 1, which means it has a good measure of separability. A poor model has an AUC near 0, which means it has the worst measure of separability.

The outcome of the TP identification in the Fast (X, Y) = (5, 5) speed bin is shown in Figure 15. The accuracy of SSTA and NGTM in detecting TPs is compared in the top row. The false positive detection rate for each model across several benchmarks is shown. To forecast the change in slack, this figure compares the performance of the Stacked-regression and MLP-regression models for HT detection. Five different iterations of the NGTM model (NGTM-X) are provided, with each iteration having been trained with X HT included in its training set, where X∈[0,1,5,10,15]. As reported, the inclusion of a small number of HT samples in our data set minimally impacts the detection rate of ML-HTD (using NGTM) on the test set, as the detection rate and false-positive rate of ML-HTD for NGTM-0 is similar to the NGTM-X for X∈[0,1,5,10,15]. The similarity of detection rate and false-positive rate is simply because the number of HT is not statistically significant to affect the training (e.g., 15 HT data versus 20K HT free data points) process.

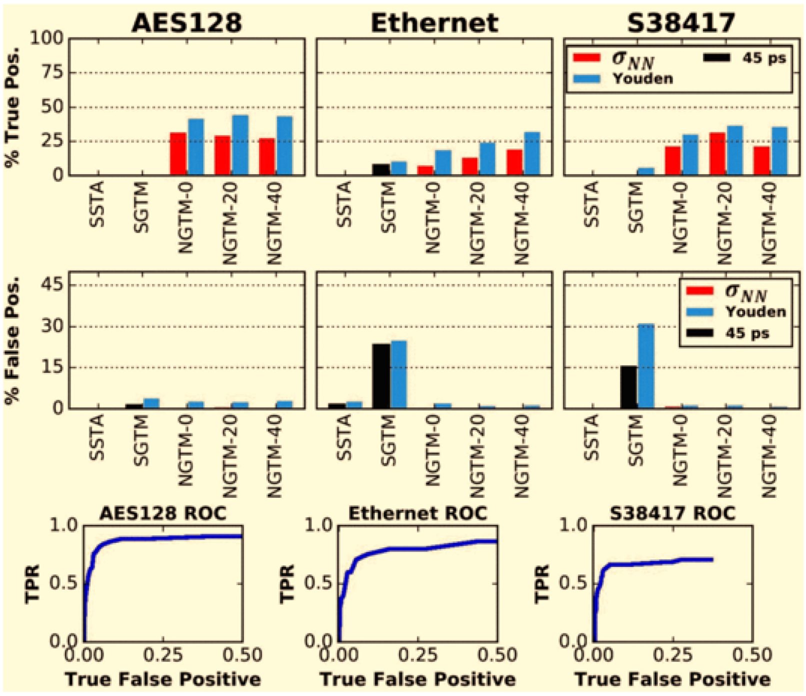

Fig. 16. Trojan trigger detection results.

Since the detection rate and false-positive rate of ML-HTD for NGTM-0 and NGTM-X for X[0,1,5,10,15] are comparable, the inclusion of a few HT samples in our data set had a negligible effect on the rate of ML-HTD detection on the test set (using NGTM). The closeness in the detection and false-positive rates is because the number of HT (e.g., 15 HT data versus 20K HT-free data points) is not statistically significant enough to influence the training process.

The outcome of our TT identification in the FAST speed bin with (X, X) = (5, 5) is shown in Figure 16. It contrasts the efficiency of SSTA and various types of NGTM (Trojan-tainted model) for identifying TTs, much like the TP scenario. The graph illustrates how the detection rate for TTs is greatly improved by a decrease in the NN-standard watchdog's deviation (over 40% in some cases). As demonstrated, NGTM has a lower rate of identifying TTs than TPs because TT has a lesser effect on the latency of impacted observed nets than TP (which is at least equal to one gate delay). We see that, similar to the TP situation, the training set being contaminated by a small number of HT data points does not affect the trained NN-WatchDog accuracy. As shown, the accuracy of HT detection greatly depends on the learning model chosen (MLP vs. Stacked) for training the NN-watchdog. As shown by the stacking regression, a reduced threshold (chosen based on 4x(sigma NN) of regression model error) might increase the detection rate by 10% to 15%, yielding an HT detection rate above 95%. This occurs when the entire solution's false-positive rate, as determined by the stacking regression model, is lower or equal to its MLP-based equivalent. Although using the Youden threshold for detection considerably increases TT detection, it produces more false positives and might not be the best method for determining the detection threshold.

Figure 15 illustrates the ROC curve from which the Youden threshold is extracted for NGTM-10. The Youden value for other NGTM models is extracted using similar ROC curves. Table 4 compares the threshold values obtained from using the Youden method with a threshold of $4\sigma_{NN}$ of regression model error. For having a different ROC curve, the threshold obtained from the Youden method is different for TT and TP detection. However, the $4\sigma_{NN}$ threshold is fixed for both TT and TP (because the same NN for detection of TT and TP is used). As illustrated, the Youden threshold is smaller (for both TT and TP detection) than the threshold. This explains why HT detection using the Youden threshold results in a higher detection rate in Figures 15 and 16 for TT and TP detection. But, as illustrated in these figures, the smaller threshold comes at the expense of a significantly higher false-positive rate. This table also compares the threshold values obtained using the stacking learning solution and when using an MLP solution.

The TTs may be engineered to have a negligible delay impact on the affected timing paths, making them more covert. A tiny change in the delay of the affected timing paths might be achieved by connecting TTs to gates with high drive strength and low threshold voltage and minimizing their capacitive delay by shortening the TT nets. However, as mentioned in section IV-C, by capping the size of standard cells utilized in the design, boosting usage in regions that need to be safeguarded, and mandating high routing density across those areas, we may make the system more sensitive to Trojan Triggers. This would force the adversary to connect the TT to cells with smaller drive strength and use longer nets to connect the Trojan Trigger to the Trojan logic (TT placed further away). An experimental SPICE framework is set up for evaluating how a Trojan Trigger affects delay at various distances from its driving cell. For a process with 32nm technology, metal 3, we employed a distributed RC model. The effect of increasing the TT distance (and associated capacitive delay) on the latency of a timing path built using five NAND gates have been modeled. Assuming our detection threshold is 25ps, the timing-path delay will rise by 25ps when the TT introduces an extra capacitance equivalent to a net that drives the TT logic positioned 40 m distant from the impacted net, as shown. Sensitization may therefore be a successful strategy for raising the HT detection rate.

6 FUTURE DIRECTIONS AND DISCUSSION

In this section, we will discuss the future directions for ML-based HT detection. State-of-the-art techniques and enablers for HT detection will also be discussed.

Some of the drawbacks with HT detection methods, in general, are automation of the HT detection flow, integration of state-of-the-art ML, adoption of emerging imaging, testing, and side-channel techniques to extract IC features, and effective HT insertion methods to validate HT detection schemes. Automating the EDA tool flow, ML framework, and analytics for HT detection is an important task. A typical HT detection methodology may consist of several EDA tools and ML frameworks at different abstraction layers. To aid in HT detection, automation scripts and flows are required to integrate multiple EDA tools and ML frameworks. In the ML-HTD example explained in this tutorial, we have used several EDA tools for feature extraction, HT insertion, netlist extraction, physical design, sign-off, and GDSII generation. To make it easier for the end-user to detect HTs in ICs, it is crucial to develop automated HT detection tools and frameworks. Improvements in HT insertion strategies drive the advancement of HT detection. An efficient HT insertion scheme provides a better challenge and feedback for the HT detection scheme being tested. In [56], researchers used Reinforcement Learning for HT insertion. Their insertion mechanism is based on the Proximal Policy Optimization algorithm, and they randomly insert HTs in the design. However, In their insertion mechanism, they have not considered the design features. The main point in using RL is problem size and search space that here depend on the size of the circuit. So, considering that, one of the promising directions could be RL-based HT insertion based on the design features. So, this way, the insertion mechanism would be different for each design and specific to that design. Another possibility is having a completely automated HT validation tool that would be helpful for academic research. In a way, we give the design netlist as input to the tool, and it inserts the HT into the design. Also, it has the capability of supporting different ML-based HT detection approaches.

Advancements in imaging, testing, and IC data extraction techniques have been key enablers for reverse engineering and, as a result, HT detection. Authors in [57] demonstrated the first use of high spatial resolution and wide field-of-view magnetic field measurements using the **Quantum Diamond Microscope (QDM)** for HT detection. They also showed that this side channel data could be used for HT detection through a Convolutional Neural Network (CNN) and clustering analysis for unsupervised deep learning. Researchers in [58] have proposed **ptychographic X-ray computed tomography (PXCT)**, where an IC's intricate interconnect network is mapped onto a 3D canvas.

After mounting the sample on a mechanical stage that allowed them to rotate it along its cylindrical axis, they shot an X-ray beam through the side of the sample. The IC sample was then lit with a sequence of overlapping 2m wide dots as it rotated. The coherent X-rays diffracted as they traveled through the chip's intricate network of copper interconnects at each lighted point, producing a pattern onto a detector that was then saved for further processing. The three-dimensional structure might be identified from the recorded projections' information about the substance the X-rays passed through. The computational method of ptychography creates an image of a specimen from the light's interference pattern passing through it. The experiment in the PXCT [58] was conducted on an Intel Pentium G3260 manufactured using 22nm FinFET technology.

State-of-the-art HTs are stealthy, and adversaries are finding newer strategies to insert and hide malicious HTs at different stages of the IC design and supply chain. One way of hiding HTs from image-based detection methods is by using reinforcement learning to iteratively find the best possible location and position to hide HTs and validate the placement using detection schemes. Another area of HT detection to be investigated is the adversarial insertion of HT in the HT detection framework. Adversarial insertion of HTs in HT detection tools or frameworks can fool the ML-based HT detection model into thinking there is no Trojan while a Trojan is inserted. Tricking the testing team by making an IC look like a Trojan-free IC while an HT is inserted, is far more malicious once there is some confidence that the IC is Trojan-free, as this IC may be used in highly confidential or critical applications given that there is an all clear from the Trojan detection model. Researchers are investigating adversarial attacks on deep learning models and also a possible solution to keep the model secure against state-of-the-art adversarial attacks [59–63]

Another question is if we can use GANs for Trojan insertion for better IC Trojan detection validation [30]. Another direction is exploiting the NLP-based model to detect Trojans. For example, we can use BILSTM models for processing circuit timing features similar to this work [64] and use that for Trojan detection. Another option could be the use of different Transformer based models [65] that are powerful NLP tools to process design features and predict the possibility of having a Trojan-free circuit with more accuracy. Taking a holistic approach to mitigating hardware security threats, the threat of HTs, in this case, is crucial to the development and security of an organization and nation.

7 CONCLUSION

In this tutorial, we provided a comprehensive overview of ML-based HT detection methods and illustrated with an example, ML-Assisted HT Detection (ML-HTD). ML-Assisted HT Detection (ML-HTD) is a methodology for Trojan detection that builds over LASCA [66]. ML-HTD does not require a Golden IC but relies on two factors: improving the timing model at design time to account for voltage noise, and training a Neural Network, which is used as a process tracking watchdog, at test time to model the process drift while accounting for process variations. The complete Trojan detection flow and ML framework are explained in detail, along with a brief explanation of individual ML models. Detection results are reported, and evaluation metrics are explained. Finally, we review the trends, future challenges, and outlook of using ML methods for HT detection.

REFERENCES

[1] Srivaths Ravi, Anand Raghunathan, Paul Kocher, and Sunil Hattangady. 2004. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)* 3, 3 (2004), 461–491.

[2] Seetharam Narasimhan, Dongdong Du, Rajat Subhra Chakraborty, Somnath Paul, Francis G. Wolff, Christos A. Papachristou, Kaushik Roy, and Swarup Bhunia. 2012. Hardware Trojan detection by multiple-parameter side-channel analysis. *IEEE Transactions on Computers* 62, 11 (2012), 2183–2195.

[3] Reza Rad, Jim Plusquellic, and Mohammad Tehranipoor. 2008. Sensitivity analysis to hardware Trojans using power supply transient signals. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 3–7.

[4] Dongdong Du, Seetharam Narasimhan, Rajat Subhra Chakraborty, and Swarup Bhunia. 2010. Self-referencing: A scalable side-channel approach for hardware Trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 173–187.

[5] Domenic Forte, Chongxi Bao, and Ankur Srivastava. 2013. Temperature tracking: An innovative run-time approach for hardware Trojan detection. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 532–539. DOI: http://dx.doi.org/10.1109/ICCAD.2013.6691167

[6] Yu Liu, Yier Jin, and Yiorgos Makris. 2013. Hardware Trojans in wireless cryptographic ICs: Silicon demonstration & detection method evaluation. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13)*. 399–404. DOI: http://dx.doi.org/10.1109/ICCAD.2013.6691149

[7] Kan Xiao, Xuehui Zhang, and Mohammad Tehranipoor. 2013. A clock sweeping technique for detecting hardware Trojans impacting circuits delay. *IEEE Design & Test* 30, 2 (2013), 26–34.

[8] Yier Jin and Yiorgos Makris. 2008. Hardware Trojan detection using path delay fingerprint. In *2008 IEEE International Workshop on Hardware-oriented Security and Trust*. IEEE, 51–57.

[9] Jie Li and John Lach. 2008. At-speed delay characterization for IC authentication and Trojan horse detection. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 8–14.

[10] Xiaotong Cui, Elnaz Koopahi, Kaijie Wu, and Ramesh Karri. 2018. Hardware Trojan detection using the order of path delay. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14, 3 (2018), 1–23.

[11] Ingrid Exurville, Loic Zussa, Jean-Baptiste Rigaud, and Bruno Robisson. 2015. Resilient hardware Trojans detection based on path delay measurements. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST'15)*. IEEE, 151–156.

[12] Dylan Ismari, Jim Plusquellic, Charles Lamech, Swarup Bhunia, and Fareena Saqib. 2016. On detecting delay anomalies introduced by hardware Trojans. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'16)*. ACM, 1–7.

[13] Yu Liu, Ke Huang, and Yiorgos Makris. 2014. Hardware Trojan detection through golden chip-free statistical side-channel fingerprinting. In *Proceedings of the 51st Annual Design Automation Conference*. 1–6.

[14] Ashkan Vakil, Ali Mirzaeian, Houman Homayoun, Naghmeh Karimi, and Avesta Sasan. 2021. AVATAR: NN-assisted variation aware timing analysis and reporting for hardware Trojan detection. *IEEE Access* 9 (2021), 92881–92900.

[15] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. 2013. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st International Conference on Computer Design (ICCD'13)*. IEEE, 471–474.

[16] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. 2017. Benchmarking of hardware Trojans and maliciously affected circuits. *Journal of Hardware and Systems Security* 1, 1 (2017), 85–102.

[17] Karim Arabi, Resve Saleh, and Xiongfei Meng. 2007. Power supply noise in SoCs: Metrics, management, and measurement. *IEEE Design & Test of Computers* 24, 3 (2007), 236–244.

[18] Kento Hasegawa, Masao Yanagisawa, and Nozomu Togawa. 2017. A hardware-Trojan classification method using machine learning at gate-level netlists based on Trojan features. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 100, 7 (2017), 1427–1438.

[19] Kevin Immanuel Gubbi, Sayed Aresh Beheshti-Shirazi, Tyler Sheaves, Soheil Salehi, Sai Manoj PD, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. 2022. Survey of machine learning for electronic design automation. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 513–518.

[20] Kento Hasegawa, Masao Yanagisawa, and Nozomu Togawa. 2017. Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS'17)*. IEEE, 1–4.

[21] Chongxi Bao, Domenic Forte, and Ankur Srivastava. 2014. On application of one-class SVM to reverse engineering-based hardware Trojan detection. In *Fifteenth International Symposium on Quality Electronic Design*. 47–54. DOI: http://dx.doi.org/10.1109/ISQED.2014.6783305

[22] Abdurrahman A. Nasr and Mohamed Z. Abdulmageed. 2017. An efficient reverse engineering hardware Trojan detector using histogram of oriented gradients. *Journal of Electronic Testing* 33, 1 (2017), 93–105.
[23] Chongxi Bao, Domenic Forte, and Ankur Srivastava. 2016. On reverse engineering-based hardware Trojan detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (2016), 49–57. DOI: http://dx.doi.org/10.1109/TCAD.2015.2488495
[24] Hassan Salmani. 2017. COTD: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist. *IEEE Transactions on Information Forensics and Security* 12, 2 (2017), 338–350. DOI: http://dx.doi.org/10.1109/TIFS.2016.2613842
[25] Mingfu Xue, Jian Wang, and Aiqun Hu. 2016. An enhanced classification-based golden chips-free hardware Trojan detection technique. In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST'16)*. 1–6. DOI: http://dx.doi.org/10.1109/AsianHOST.2016.7835553
[26] Ashkan Vakil, Farnaz Behnia, Ali Mirzaeian, Houman Homayoun, Naghmeh Karimi, and Avesta Sasan. 2020. LASCA: Learning assisted side channel delay analysis for hardware Trojan detection. In *2020 21st International Symposium on Quality Electronic Design (ISQED'20)*. 40–45. DOI: http://dx.doi.org/10.1109/ISQED48828.2020.9137007
[27] S. Moustakidis, K. Liakos, G. Georgakilas, Nikolaos Sketopoulos, Stavros Seimoglou, Patrik Karlsson, and Fotios Plessas. 2020. A novel holistic approach for hardware Trojan detection powered by deep learning (HERO). In *Proc. ATTRACT'20*.
[28] Rozhin Yasaei, Luke Chen, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. 2022. Hardware Trojan detection using graph neural networks. *arXiv preprint arXiv:2204.11431* (2022).
[29] Rozhin Yasaei, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. 2021. GNN4TJ: Graph neural networks for hardware Trojan detection at register transfer level. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE'21)*. IEEE, 1504–1509.
[30] Fredin Jose, M. Priyatharishini, and M. Nirmala Devi. 2022. Hardware Trojan detection using deep learning-generative adversarial network and stacked auto encoder neural networks. In *ICT Analysis and Applications*. Springer, 203–210.
[31] Huili Chen, Xinqiao Zhang, Ke Huang, and Farinaz Koushanfar. 2022. AdaTest: Reinforcement learning and adaptive sampling for on-chip hardware Trojan detection. *arXiv preprint arXiv:2204.06117* (2022).
[32] Nicholas Tuzzio, Kan Xiao, Xuehui Zhang, and Mohammad Tehranipoor. 2012. A zero-overhead IC identification technique using clock sweeping and path delay analysis. In *Proceedings of the Great Lakes Symposium on VLSI*. 95–98.
[33] Ashkan Vakil, Houman Homayoun, and Avesta Sasan. 2019. IR-ATA: IR annotated timing analysis, a flow for closing the loop between PDN design, IR analysis & timing closure. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 152–159.
[34] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. 2007. Trojan detection using IC fingerprinting. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 296–310.
[35] Reza Rad, Jim Plusquellic, and Mohammad Tehranipoor. 2009. A sensitivity analysis of power signal methods for detecting hardware Trojans under real process and environmental conditions. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 18, 12 (2009), 1735–1744.
[36] Hassan Salmani, Mohammad Tehranipoor, and Jim Plusquellic. 2009. New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 66–73.
[37] Sheng Wei and Miodrag Potkonjak. 2011. Scalable hardware Trojan diagnosis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20, 6 (2011), 1049–1057.
[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[39] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. 2015. XGBoost: Extreme gradient boosting. *R Package Version 0.4-2* 1, 4 (2015), 1–4.
[40] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.
[41] Zheng Zhang, Zhihui Lai, Yong Xu, Ling Shao, Jian Wu, and Guo-Sen Xie. 2017. Discriminative elastic-net regularized linear regression. *IEEE Transactions on Image Processing* 26, 3 (2017), 1466–1481.
[42] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 2 (2005), 301–320.
[43] Hui Zou and Hao Helen Zhang. 2009. On the adaptive elastic-net with a diverging number of parameters. *Annals of Statistics* 37, 4 (2009), 1733.
[44] J. Ranstam and J. A. Cook. 2018. LASSO regression. *Journal of British Surgery* 105, 10 (2018), 1348–1348.
[45] Lukas Meier, Sara Van De Geer, and Peter Bühlmann. 2008. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70, 1 (2008), 53–71.
[46] Gary C. McDonald. 2009. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 1 (2009), 93–100.

[47] Leonardo Noriega. 2005. Multilayer perceptron tutorial. *School of Computing. Staffordshire University* (2005).

[48] Mariana Belgiu and Lucian Drăguţ. 2016. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing* 114 (2016), 24–31.

[49] Gérard Biau and Erwan Scornet. 2016. A random forest guided tour. *Test* 25, 2 (2016), 197–227.

[50] Abdullah Nazma Nowroz, Kangqiao Hu, Farinaz Koushanfar, and Sherief Reda. 2014. Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 12 (2014), 1792–1805.

[51] Roshni Shende and Dayanand D. Ambawade. 2016. A side channel based power analysis technique for hardware Trojan detection using statistical learning approach. In *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN'16)*. IEEE, 1–4.

[52] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[53] Sebastian Raschka. 2018. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *The Journal of Open Source Software* 3, 24 (April 2018). DOI: http://dx.doi.org/10.21105/joss.00638

[54] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis (IWLS):* http://www.iwls.org.

[55] Neil J. Perkins and Enrique F. Schisterman. 2006. The inconsistency of "optimal" cutpoints obtained using two criteria based on the receiver operating characteristic curve. *American Journal of Epidemiology* 163, 7 (2006), 670–675.

[56] Amin Sarihi, Ahmad Patooghy, Peter Jamieson, and Abdel-Hameed A. Badawy. 2022. Hardware Trojan insertion using reinforcement learning. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 139–142.

[57] Maitreyi Ashok, Matthew J. Turner, Ronald L. Walsworth, Edlyn V. Levine, and Anantha P. Chandrakasan. 2022. Hardware Trojan detection using unsupervised deep learning on quantum diamond microscope magnetic field images. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* (2022).

[58] Anthony F. J. Levi and Gabriel Aeppli. 2022. The naked chip: No trade secret or hardware Trojan can hide from ptychographic X-ray laminography. *IEEE Spectrum* 59, 5 (2022), 38–43.

[59] Johann Knechtel, Elif Bilge Kavun, Francesco Regazzoni, Annelie Heuser, Anupam Chattopadhyay, Debdeep Mukhopadhyay, Soumyajit Dey, Yunsi Fei, Yaacov Belenky, Itamar Levi, et al. 2020. Towards secure composition of integrated circuits and electronic systems: On the role of EDA. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*. IEEE, 508–513.

[60] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284* (2017).

[61] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

[62] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. 2018. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9185–9193.

[63] Naveed Akhtar and Ajmal Mian. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access* 6 (2018), 14410–14430.

[64] Tanmoy Chowdhury, Ashkan Vakil, Banafsheh Saber Latibari, Seyed Aresh Beheshti Shirazi, Ali Mirzaeian, Xiaojie Guo, Sai Manoj P. D., Houman Homayoun, Ioannis Savidis, Liang Zhao, et al. 2022. RAPTA: A hierarchical representation learning solution for real-time prediction of path-based static timing analysis. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 493–500.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[66] Ashkan Vakil, Farnaz Behnia, Ali Mirzaeian, Houman Homayoun, Naghmeh Karimi, and Avesta Sasan. 2020. LASCA: Learning assisted side channel delay analysis for hardware Trojan detection. In *2020 21st International Symposium on Quality Electronic Design (ISQED'20)*. IEEE, 40–45.

Received 1 July 2022; revised 19 December 2022; accepted 1 January 2023