A Survey on Machine Learning in Hardware Security

Köylü, T.C.; Reinbrecht, Cezar; Gebregiorgis, A.B.; Hamdioui, S.; Taouil, M.

DOI
10.1145/3589506

Publication date
2023

Document Version
Final published version

Published in
ACM Journal on Emerging Technologies in Computing Systems

A Survey on Machine Learning in Hardware Security

TROYA ÇAĞIL KÖYLÜ, CEZAR RODOLFO WEDIG REINBRECHT, ANTENEH GEBREGIORGIS, SAID HAMDIOUI, and MOTTAQIALLAH TAOUIL,
Delft University of Technology, the Netherlands

Hardware security is currently a very influential domain, where each year countless works are published concerning attacks against hardware and countermeasures. A significant number of them use machine learning, which is proven to be very effective in other domains. This survey, as one of the early attempts, presents the usage of machine learning in hardware security in a full and organized manner. Our contributions include classification and introduction to the relevant fields of machine learning, a comprehensive and critical overview of machine learning usage in hardware security, and an investigation of the hardware attacks against machine learning (neural network) implementations.

CCS Concepts: • **Security and privacy → Hardware attacks and countermeasures**; • **Computing methodologies → Machine learning**; **Neural networks;**

Additional Key Words and Phrases: Hardware security, machine learning, hardware attacks, hardware countermeasures, neural networks, cybersecurity

ACM Reference format:
Troya Çağıl Köylü, Cezar Rodolfo Wedig Reinbrecht, Anteneh Gebregiorgis, Said Hamdioui, and Mottaqiallah Taouil. 2023. A Survey on Machine Learning in Hardware Security. *ACM J. Emerg. Technol. Comput. Syst.* 19, 2, Article 18 (May 2023), 37 pages.
https://doi.org/10.1145/3589506

1 INTRODUCTION

Ensuring the security of digital devices has become an important requirement [1]. As a result, great attention was given to cybersecurity with a focus on software [2]. It was soon realized however that attacking the hardware instead was much more rewarding, and consequently, security cannot be attained without protecting the hardware. In light of this, many researchers started using tools from other domains for improving attacks or countermeasures, as hardware security is a cat-and-mouse game between them. Machine learning (ML) has proven to be one of the most prominent such tools. This was to be expected, as machine learning has already achieved many tasks in other domains (e.g., game playing [3] and carrying out daily conversations [4]). To what effect hardware security researchers are using machine learning is however another issue. There is a need for a complete overview that highlights missed opportunities.

Authors' address: T. Ç. Köylü, C. R. Wedig Reinbrecht, A. Gebregiorgis, S. Hamdioui, and M. Taouil, Delft University of Technology, Mekelweg 4, 2628CD, Delft, South Holland, the Netherlands; emails: T.C.Koylu@tudelft.nl, C.R.WedigReinbrecht@tudelft.nl, A.B.Gebregiorgis@tudelft.nl, S.Hamdioui@tudelft.nl, M.Taouil@tudelft.nl.

1550-4832/2023/05-ART18 $15.00
https://doi.org/10.1145/3589506

The existing reviews of machine learning for hardware security generally focus on specific applications. In the works of Maghrebi et al. (2016) and Cagli et al. (2017), the authors investigated machine learning in the context of profiling-based side-channel attacks [5, 6]. Picek et al. (2018) also provided commentary in the same context [7]. Hettwer et al. (2020) expanded the coverage to investigate the machine learning usage in the entirety of side-channel attacks [8]. The work of Maes and Verbauwhede (2010) briefly mention the usage of machine learning in attacking Physically unclonable function (PUF)s, whereas Rührmair and Sölter (2014) directly focus on that topic [9, 10]. The survey of Rahman et al. (2018) dedicates a section to describe the role of machine learning in physical inspection attacks (such as reverse engineering, side channel-analysis, and fault injection) [11]. Likewise, Liu et al. (2020) dedicates a section on how approximate computing techniques can improve machine learning-based hardware attacks such as Side-channel analysis (SCA) and attacking PUFs [169]. On the countermeasure side, Huang et al. (2020) and Kundu et al. (2021) provided surveys on machine learning algorithms used for detecting Hardware Trojan (HT)s [12, 13]. The surveys that have holistic approaches are very few and they lack certain properties such as commentary on machine learning usage per case to guide future development. More importantly, they are not complete about attacks against hardware that runs machine learning (most prominently, neural networks) or countermeasures [14, 15]. This point is becoming more and more important as machine learning algorithms are being increasingly used in safety-critical applications, such as autonomous driving. Therefore, there is a need for a survey that not only is complete; but also machine learning beginner friendly, guides the reader with each application case, and presents future directions.

We address this by providing a comprehensive overview of machine learning algorithms used in hardware security, as well as hardware security issues with machine learning applications, alongside extensive commentary. In summary, the contributions of this paper are as follows:

- Providing an organized and comprehensive overview of the works that use machine learning in hardware security.
- Extending the overview to cover the hardware attacks against neural networks.
- Highlighting important points and missed opportunities about the use of machine learning.
- Providing a classification and introduction to machine learning used in hardware security.

The rest of the paper is organized as follows. Section 2 provides a functional classification and introduction to machine learning. Section 3 classifies the areas where machine learning is employed in the context of hardware security. Section 4 discusses machine learning algorithms used in hardware attacks, while Section 5 discusses machine learning algorithms used as countermeasures against hardware attacks. Then, Section 6 describes hardware-based attacks and countermeasures on neural networks. Finally, Section 7 concludes the paper by providing a summary and future insights.

2 INTRODUCTION TO MACHINE LEARNING

Machine learning aims to improve an automated data processing task with experience [16]. This experience is learned from training data, or more precisely, from the features of this data. The first step of a machine learning algorithm is therefore the *feature extraction* phase, where meaningful elements from the data are extracted. The second step is the *training* of the algorithm, which is the learning of experience (training data). The final step is the *evaluation*. In this phase, the performance of the algorithm is measured by using new data as input.

With this basis, many machine learning algorithms have been proposed. Figure 1 classifies them and indicates the most popular algorithms [16, 17] that are also used in hardware security. The first metric for classification in Figure 1 is based on the learning or *training* method. There are three

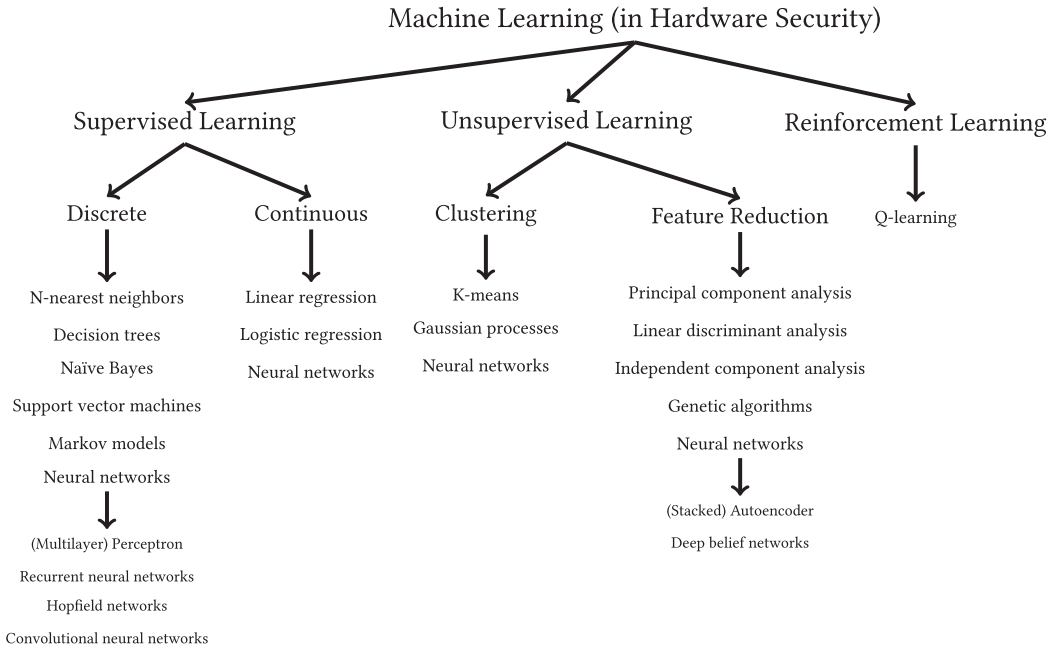

Fig. 1. Classification of machine learning methods.

training strategies: supervised, unsupervised, and reinforced. The following subsections describe these strategies, classify them further, and introduce examples of each class.

2.1 Supervised Learning

Supervised learning uses labeled data during training. This means that there is a *teacher* that provides the labels or desired states for the training data [18]. It is possible to further divide supervised learning algorithms into two sub-classes, depending on whether an algorithm produces discrete or continuous variables. The production of discrete variables is used for the classification or categorization of data. The production of continuous variables on the other hand is referred to as regression, which is typically used for data prediction or function estimation. Table 1 summarizes the examples of supervised algorithms, by also providing the data complexity they are most suitable to be used for. Each is also further described thereafter.

Discrete algorithms:

- *N-nearest neighbors* is a simple classification algorithm that classifies a data sample based on N nearest data samples. The notion of distance is attained by a metric, such as the Euclidean or Hamming distance [19]. The algorithm is suitable for non-complex classification tasks. The performance of this algorithm mostly depends on the selection of N, where a too-small number can miss the overall picture, while a too-large number would be too general for a classification. A basic example is provided in Figure 2, where the black dot is classified into the blue circle class, using $N = 3$ nearest neighbors (see the dotted circle). This is because the three nearest instances consist of two blue circles and one red cross.
- *Decision trees* are classification algorithms that build a tree-like model on data. The leaves of the decision tree are classes, and all other nodes are feature-based tests. Therefore, branches

Table 1. Supervised Machine Learning Algorithms

name	type	complexity
N-nearest neighbors	discrete	simple
Decision trees	discrete	simple
Naïve Bayes	discrete	simple & complex
SVM	discrete	complex
Markov models	discrete	complex
MLP	discrete & continuous	simple & complex
CNN	discrete & continuous	complex
Linear and logistic regression	continuous	simple

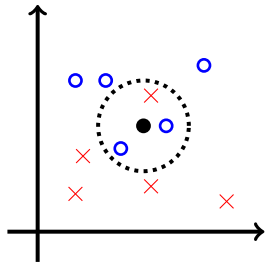

Fig. 2. *N-nearest neighbors* classification example.

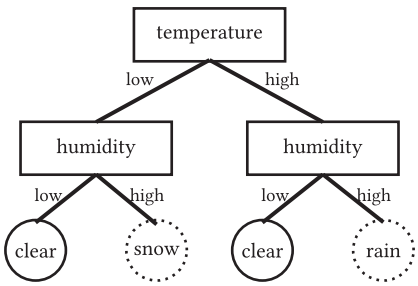

Fig. 3. *Decision tree* classification example.

connected to a node represent the possible outcomes of a feature test [20]. Decision trees are suitable to classify simple data structures, but they suffer when the data is complex. To obtain stronger classifiers, a common approach is to use multiple independent decision trees and combine the result with a majority vote. This approach is called the *decision forest*. An example of a simple decision tree is provided in Figure 3. Using this tree, a new day's weather can be classified by following the tests, based on the measurements of temperature and humidity.

- *Naïve Bayes* is a probability-based classification algorithm. The algorithm assumes that there is no correlation between the features of the data instances when the class information of the instance is known (i.e., features are independent given class). Based on this assumption, the algorithm uses the Bayes formula to determine which class that a data instance $\overline{x} = [x_0, \ldots, x_j, \ldots, x_N]$ belongs to, as follows:

$$class_x = \arg\max_i \prod_{j=1}^{n} P(x_j|C_i)P(C_i), \tag{1}$$

where C_i's are classes. While the independence assumption is rather strong and generally inaccurate, the *naïve Bayes* classifier is observed to be useful in both simple and complex data [21].

If desired, the *naïve Bayes* assumption can be replaced by a *multivariate Gaussian model* distribution, which fits certain data more accurately (it is up to the designer to determine the best assumption) [22]. The resulting classification algorithm becomes the following.

$$\begin{aligned} class_x &= \arg\max_i \text{pdf}_i(x) \\ &= \frac{1}{(2\pi)^{n/2} \mid \mathbf{\Sigma}_i \mid^{1/2}} \\ &* \exp\left(-\frac{1}{2}(\overline{x} - \overline{\mu_i})^T \mathbf{\Sigma_i}(\overline{x} - \overline{\mu_i})\right), \end{aligned} \tag{2}$$

where $\overline{\mu}$ is the mean vector, $\mathbf{\Sigma}$ is the covariance matrix, and pdf is the Probability density function. These can be calculated using the same data points.

- Support vector machine (SVM) [23] is a linear classifier that aims to linearly separate data into two different classes, by maximizing the separation as much as possible. Therefore, the separation equation that should be maximized (with adjusting w's and b) is as follows:

$$y_k[\overline{w}^T \gamma(\overline{x}_k) + b] \geq 1 - \varepsilon_k, \tag{3}$$

where $\overline{x}_k$'s are training instances and y_k's are corresponding labels. ε_k is a compromise term, that allows a less strict separation to limit outlier influence. Lastly, $\gamma(.)$ is an implicit function that maps the input to a higher dimension.

The maximization of the separation while attaining classification correctness can be addressed by the *Lagrange multipliers* method, which creates a *dual problem* in the form:

$$\begin{aligned} \max_{\alpha} \min_{w,b,\varepsilon} \mathfrak{L} &= \frac{1}{2}\overline{w}^T\overline{w} \\ &+ \sum_k \alpha_k(1 - y_k(\overline{w}^T\gamma(x_k) + b) - 1 + \varepsilon_k) \\ &- C\sum_k \varepsilon_k. \end{aligned} \tag{4}$$

Here, C is the trade-off parameter. Taking the partial derivatives of this equation (Lagrangian) and equating to zero translates the dual problem into a *quadratic programming* problem. The solution yields the following findings (as an overview):

- There is no need to select or calculate $\gamma(.)$. Rather, a kernel is selected (linear, polynomial, Radial basis function (RBF) etc.) in the form $K(x, x_k)$.
- x_k's with nonzero α_k's constitute support vectors.
- New data (z) can be classified as:

$$z = \sum_{x_k \in S} \alpha_i y_i K(z, x_k) + b. \tag{5}$$

In this equation, S is the set of support vectors.

SVMs are used in many tasks. Although they are binary classifiers, multiclass classification problems can also be addressed by SVMs [24]. Note that if the selected SVM kernels do not create an adequate separation, SVM performance becomes limited.

- Markov models [25] are stochastic models to determine the likelihood of an event, given some information about the preceding events. If the previous events can be known, the modeling is referred to as not hidden. If the previous events can only be probabilistically

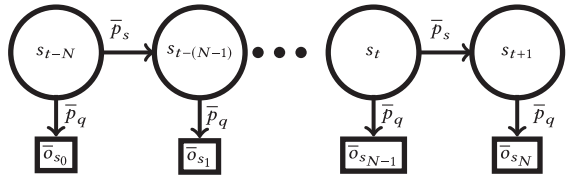

Fig. 4. Graphical representation of HMM.

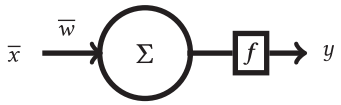

Fig. 5. The artificial neuron.

inferred via auxiliary information (called observations) rather, the model is referred to as *hidden*. Hidden Markov model (HMM)s are generally used in sound and language processing, where previous events cannot be known directly. A graphical examination is provided in Figure 4. In the figure, s denotes the states. As apparent, these are not disclosed, so the information about them comes from observations $\overline{o}$. Here, $\overline{p}_q$ denotes the probability of a certain observation to occur, given a state. Lastly, $\overline{p}_s$ denotes the probability of state transitions.
In light of this, the three main components that construct an HMM are the initial state condition probabilities ($\overline{\pi}$), observation matrix ($\boldsymbol{O}$), and the state transition matrix ($\boldsymbol{A}$); which contain all the aforementioned probabilities. When these elements are known or estimated, a number of questions can be answered by an HMM, where one of the most relevant for tasks like sound processing is "What is the most likely sequence of states given the observations?". A solution for this problem is provided by the well-known iterative Viterbi algorithm [26].

- Artificial neural network (ANN)s are complex structures based on the artificial neuron of the McCulloch-Pitts model [27], illustrated in Figure 5. This basic representation shows the processing of the input vector $\overline{x}$. First, its dot product is taken with the weight vector $\overline{w}$. The result is provided to a non-linear function f (e.g., Sigmoid, Hyperbolic tangent (tanh), Rectified linear function (ReLU)). This function produces the output y. In a biological context, this model imitates the nonlinear firing behavior of neurons. The learning is achieved by modifying $\overline{w}$ with training data.
  This neuron is used as a basis and many of them are connected with each other to form an ANN. With appropriate architectures and training strategies, many tasks have been achieved by using ANNs. Some examples of ANNs are given in the following.
- Multilayer perceptron (MLP) (see Figure 6 for a typical example) is an ANN that is composed of multiple layers of neurons: a neuron in one layer is connected to all of the neurons in the previous and the next layer, but is not connected with the neurons in the same layer. Most typically, an MLP is:
  – Consisting of an input, multiple hidden, and an output layer.
  – Used for supervised classification.
  – The number of output neurons is equal to the number of classification classes.
  – Trained by the backpropagation method [28].
  The backpropagation method aims to adjust the weights of the neurons in the network. This is achieved by first calculating the error between the input and the desired output via a loss function, during the *training* phase. The contribution to this error from individual weights is limited by adjusting them in the direction of the negative error gradient. As

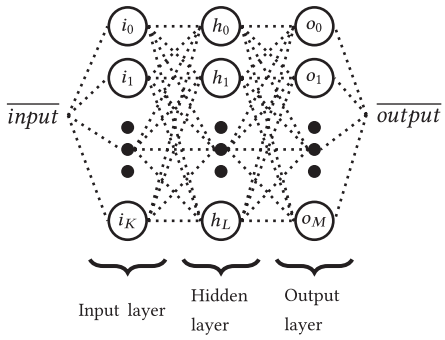

Fig. 6. A typical Multilayer perceptron.

this process is propagated from the outer layers to the inner ones, the learning method is referred to as backpropagation.

- Recurrent neural network (RNN) [29] is a neural network with "memory". This is attained with feedback loops on neurons. With these feedback loops, the previous decision affects the next operation. RNNs are appropriately used for tasks that are sequential in nature, such as speech, text, and the like.
  RNN efficiency can drop when the task requires a lot of time steps back in time. Many improvements were proposed to address this issue, with the most popular one being the Long short-term memory (LSTM). This architecture augments RNN by introducing specialized gates to process the flow of data [30].
- Hopfield networks [31] are memory recall structures. In essence, when a pattern is provided to this network, it tries to associate it with the closest learned pattern. It does this iteratively: at each iteration, this network updates the bits of the input, until two subsequent states are equal. The bitwise update formula for the modern Hopfield network is provided in the following [32].

$$\xi_{t+1}[l] = \operatorname{sgn}\left[\sum_{i=0}^{N-1} F\left(x_i^T \xi_t^{(l^+)}\right) - \sum_{i=0}^{N-1} F\left(x_i^T \xi_t^{(l^-)}\right)\right]. \tag{6}$$

  Here, $\xi_t[l]$ is the lth bit of the state at time t (where the state is equal to the input at $t = 0$) and $x_i \in [0, N)$ are learned patterns. Furthermore, $\xi_t^{(l^+)}$ and $\xi_t^{(l^+)}$ only differ at bit l, where $\xi_t^{(l^+)}[l] = 1$ and $\xi_t^{(l^-)}[l] = -1$. In this equation, the function F determines how many patterns the network can learn and how accurately it can recall them. It is common to select it as a polynomial (e.g., $\mathrm{F}(a) = a^3$) or exponential (i.e., $\mathrm{F}(a) = \exp(a)$).
  While they are not used as prominently as other neural networks, (modern) Hopfield networks can accomplish many tasks. Most commonly, they are used in image-processing tasks, such as reconstructing images with distortions [33] or generating images from text descriptions [34].
- Convolutional neural network (CNN) [35] is an ANN with a higher number of layers (thus, the association with the term *deep learning*). Some layers are specialized for visual tasks, such as image classification and object recognition. A CNN includes a couple of fundamental layer types (typically in the order presented): convolutional, pooling, nonlinearity, and fully connected/dense.
  - *Convolutional layer:* This layer can be inspected in two aspects: functional and structural. Functionally, this layer accomplishes filtering between the input and its learned filters

(different kernels), and as filtering refers to convolution, it is called the convolutional layer. The functional output of this layer is feature maps that the proceeding layers can learn from. Structurally, this layer is formed by a notion called *weight sharing*. In this layer, different neurons use the same weights to simulate a convolution. When the weight updates are calculated in the training phase, these updates are aggregated and applied the same for the shared weights.

- *Pooling layer:* This layer only compresses the output of the earlier layers. Over a grid of selected size (e.g., 4×4), it averages or takes the maximum value. So the grid is expressed by a single number. Note that there is no learning in this layer.
- *Nonlinearity layer:* Simulating the nonlinear firing behavior of a biological neuron, this layer maps its input to an output based on the selected nonlinear function. These are commonly Sigmoid, ReLU, or tanh. There is no learning also in this layer.
- *Fully connected layer:* This layer is the standard MLP layer. Found in the last part of a CNN, this structure accomplishes the final classification. An important observation here is that, due to the earlier layers (especially the pooling layer), the input to this layer has a much smaller dimension than its original form. This effective feature reduction is the key point of effectiveness of CNNs against issues like the vanishing gradient in deep neural architectures [36].

Continuous algorithms:

- Linear [37] and Logistic regression (LR) [38] are two regression methods. They both aim to explain the relation between input and output of the form $f(x) = y$, by fitting a line or curve using $\{x, y\}$ pairs in the training data. The main difference between them is what they use to fit. The fitting function that the linear regression uses is given in the following:

$$y = \alpha x + \beta. \tag{7}$$

Whereas the logistic regression curve is;

$$y = \frac{1}{1 + e^{-(\alpha+\beta x)}}. \tag{8}$$

For a better understanding, these techniques can be thought of as a single neuron, where the nonlinear function is replaced with either a linear or logistic function. The *training* procedure sets the values of α's and β's. For this, gradient descent can be used: it is not called backpropagation as there are no previous layers. Lastly, the limitations of this technique are analogous to using few neurons in an MLP. If the data is complex, the performance is poor.

2.2 Unsupervised Learning

Unsupervised learning uses unlabeled data in contrast to supervised learning. This means such machine learning algorithms should directly decide to use the properties of the provided data without the existence of a "teacher" [39]. For this reason, there is no great distinction between *training* and *evaluation* phases in unsupervised learning. It is possible to subdivide these kinds of algorithms into two: clustering and feature reduction. Clustering is the task of categorization without known labels and feature reduction is the task of reducing the dimension of the data for more effective and efficient processing. Its well-known methods are summarized in Table 2 and detailed in the following.

- K-means clustering [40] is a two-step iterative clustering algorithm. In the first step, K cluster centers are selected such that they minimize the Euclidean distance/sum of squares within a cluster. In the second step, the K clusters are reformed by assigning the closest instances to

Table 2. Unsupervised Machine Learning Algorithms

name	type	complexity
K-means	clustering	simple
Gaussian processes	clustering	simple
NN	clustering	simple & complex
PCA	feature reduction	simple & complex
Genetic algorithms	feature reduction	simple & complex
(Stacked) Autoencoder	feature reduction	complex
Deep belief network	feature reduction	complex

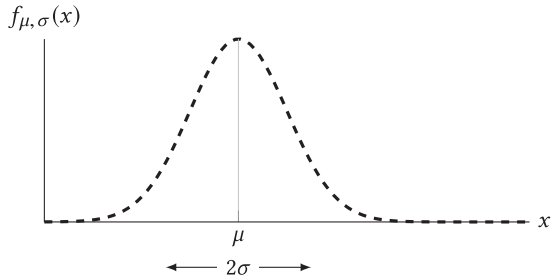

Fig. 7. Sample univariate Gaussian distribution.

the newly selected centers. The iterations continue until an adequately small sum-of-squares value is obtained, or the change between iterations is very small. The end product of this algorithm is data that is separated over K clusters. The limitations of this method arise when the processed data cannot be characterized by their average.

- Gaussian processes [41] assume that the investigated data follows one or a combination of Gaussian distribution(s) (here, we refer to using the Gaussian distribution for machine learning tasks as Gaussian processes). A simple example of data with one dimension (univariate) is illustrated in Figure 7. In the figure, the curve shows the ratio of data instances that fall under it. As can be seen in the figure, most of these instances fall near the mean μ, with a spread of standard deviation σ, two parameters that characterize a Gaussian. More complex Gaussians can be attained by considering multidimensional data (multivariate Gaussian) or assuming that the data is distributed by superimposed Gaussians rather than one (Gaussian mixture). The calculation of these parameters was already discussed in naïve Bayes.
  Although the assumption that a collection of data is distributed by a Gaussian will not perfectly hold most of the time, still, this modeling can be used for clustering. This is done by first calculating the mean and the standard deviation of the data. When a new data instance arrives, the probability that this instance(x) belongs to this distribution is found using the pdf formula $f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{-0.5(\frac{x-\mu}{\sigma})^2\}$. This enables the assignment of data points to different classes of data.
- Principal component analysis (PCA) [42] is the task of projecting multidimensional data into a smaller dimension, where it is easier to process or separate. The aim is to find orthogonal components that the data has the most variance. To this end, the calculation of the projection (vectors) includes the calculation of eigenvectors of the covariance matrix across dimensions. A couple of these eigenvectors are then selected, which correspond to the highest eigenvalues. This number of selections determines the new reduced number of

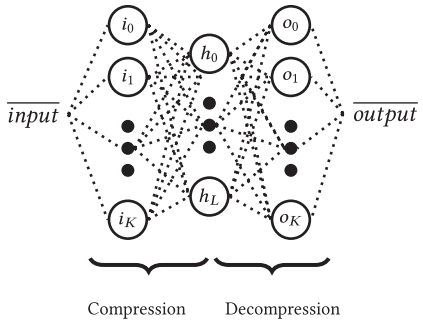

Fig. 8. A basic autoencoder.

dimensions. The projection is achieved by taking the dot product between the original data and calculated projection vectors.

- Genetic algorithms [43] are based on evolution and natural selection. Genetic algorithms for feature reduction comprise of the following steps, given a dataset that each instance contains multiple features and an assessment criterion like classification accuracy.
  - Construct an initial population, i.e., each member of the population contains which data features to select and which not to select.
  - Assess their *fitness*, i.e., determine which feature selections lead to better performance. This performance assessment can be done in a multitude of ways, where some can be supervised and some unsupervised.
  - Select the best-performing feature selection instances for future generations, i.e., *natural selection*.
  - Apply mutations to and cross-overs between the selected instances and add the new feature selections to the set.
  - Repeat the steps except the first one until a stopping criterion is met.

  The stopping criteria can for example be reducing the number of features to a certain value without sacrificing too much performance.
- Autoencoders [44] are neural networks that aim to first encode and then decode data. The aim is to minimize the difference between the original data and the decoded version. This compressed version is reduced in the number of features and can be used in other machine learning tasks. This is illustrated in Figure 8, where $L < K$. Common autoencoder usage includes using many layers for compression and decompression to obtain a deep-stacked autoencoder, adding noise to input data, and evaluating performance for a denoised decompression.
- Deep belief network (DBN)s [45] appear exactly the same as MLPs architecturally. Functionally, however, they are formed by connecting multiple Restricted boltzmann machine (RBM)s sequentially. Each RBM is a two-layered network that is trained in an unsupervised manner to translate (forward pass) and reconstruct (backward pass) its input. After this layer-by-layer training, DBN is complemented by a fully connected output layer, which is fine-tuned (training of only certain layers) to associate inputs to labels.

2.3 Reinforcement Learning

Reinforcement learning can be considered as a middle ground between supervised and unsupervised learning. It tackles the problem of being able to learn an optimal-like behavior as an agent via trial-and-error, in a dynamic environment [46]. This type of learning is especially used in

Table 3. Reinforcement Learning
Machine Learning Algorithm

name	type	complexity
Q-learning	–	complex

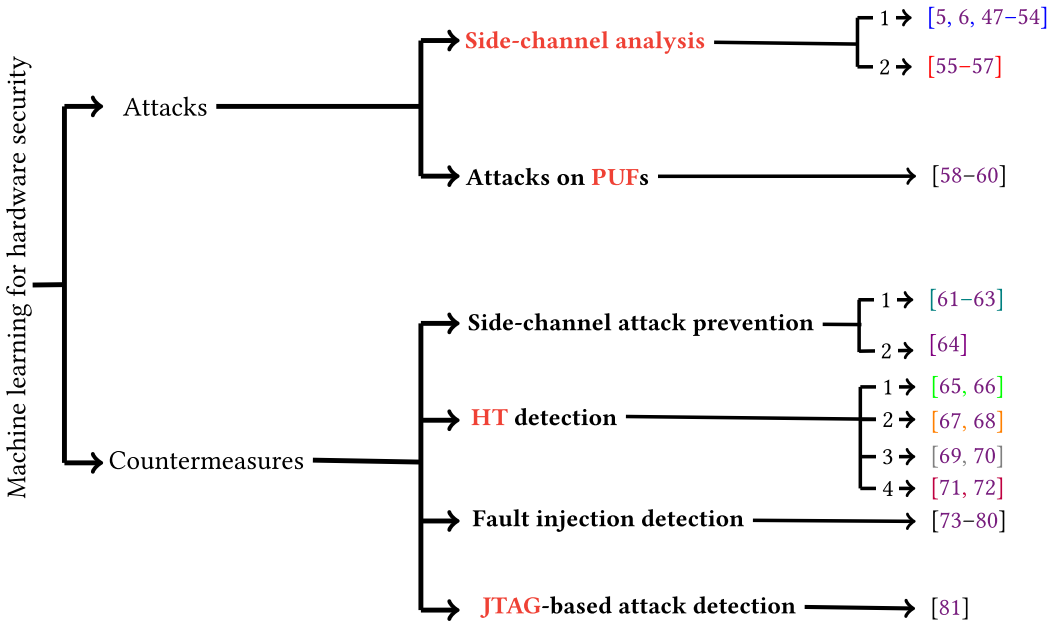

Fig. 9. Classification hardware security based on machine learning usage.

game-playing (such as backgammon or checkers) and in robot-environment interaction scenarios. Its well-known method is indicated in Table 3 and detailed in the following.

- Q-learning provides an algorithm for the question of determining the optimal set of actions in a particular environment. The methodology includes a Q-function (also referred to as the Bellman Equation) that defines the reward of an action in a state as the sum of the immediate reward and the cumulative reward of following the optimal actions after the current action. Regarding this formulation, it provides a learning algorithm that learns the values required by the Q-function. This corresponds to the finding of the optimal strategy [16].

3 CLASSIFICATION OF MACHINE LEARNING FOR HARDWARE SECURITY

In this survey, we investigate the studies that use machine learning for hardware security in two groups: hardware attacks and countermeasures. This is illustrated in Figure 9. The attacks use machine learning as an analysis tool to obtain a secret from a target device. The first phase of such an attack is the data collection from the device. This data is modeled *offline*, using a machine learning algorithm. Examples that use this methodology include side-channel analysis (Section 4.1) and attacks against PUFs (Section 4.2). Side-channel analysis can be further subdivided into two: power analysis (indicated by 1 on Figure 9) and sound analysis (2). Section 4 elaborates on these attacks.

The second group consists of countermeasures against hardware attacks. The use of machine learning for countermeasures is more varying compared to attacks; from offline tools to *online*

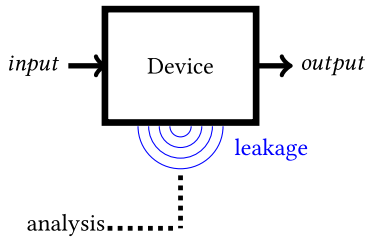

Fig. 10. A general side-channel analysis setup.

Table 4. Summary of Machine Learning-based SCA Studies

Study	SCA Method	aim	ML tool
[48]	power (1)	key leak	SVM
[49]	power (1)	key leak	MLP
[50]	power (1)	mask removal	ANN
[5]	power (1)	key leak	CNN
[6]	power (1)	key leak	CNN
[51]	power (1)	key leak	autoencoder + CNN
[52]	power (1)	key leak	CNN, MLP, LSTM
[53]	power (1)	key leak	CNN, MLP
[54]	power (1)	key leak	CNN
[83]	power (1)	key leak	MLP
[55]	sound (2)	typing reconstruction	MLP
[56]	sound (2)	typing reconstruction	k-means + HMM, LDA, Gaussian process, MLP
[57]	sound (2)	printing reconstruction	HMM

detectors (i.e., a machine learning algorithm constantly working to detect attacks). Example countermeasures include side-channel analysis prevention, HTs, fault injection, and JTAG-based attack detection. Side-channel analysis prevention can be investigated in two groups as active (1) and passive (2). We can likewise investigate HT detection in four groups: reference-based detection (1), reference-free detection with side-channel fingerprinting (2), reference-free detection using image or signal properties (3), and reference-free detection using graph theory (4). Section 5 discusses these countermeasures further.

4 MACHINE LEARNING BASED HARDWARE ATTACKS

In this section, we first present side-channel attacks in Section 4.1. Next, we present the attacks on PUFs in Section 4.2.

4.1 Side-channel analysis

SCA exploits physical characteristics of a device (e.g., power consumption, sound emission, temperature) to obtain a secret [82] as illustrated in Figure 10. There are two main use cases of machine learning tools in the context of side-channel analysis in the literature. The vast majority of studies focus on breaking cryptographic algorithms by analyzing the power consumption of the device, while there are some studies that try to reconstruct information using the emitted sound.

These studies are summarized in Table 4 and discussed in detail in two subsections next.

4.1.1 Power (Side-channel) Analysis. Power analysis aims to obtain a secret by modeling the power consumption of the device. In order to be clear in our explanation, we first provide background information about the power analysis techniques that led to the usage of machine learning.

Background on power analysis. A first power analysis study is presented by Kocher et al. (1999). In this study, the authors presented two approaches: simple and differential fault analysis [84].

Simple power analysis (SPA) is based on the direct interpretation of the recorded power trace. A common example is the power analysis of the Square-and-multiply (SaM) algorithm. This iterative algorithm squares a number in each iteration and makes an additional multiplication based on the bit in consideration. As the additional multiplication uses more power, it is possible to decode the key bits at each iteration. However, this decoding is only possible when the trace is not overshadowed by noise.

To consider the noisy case, the authors presented Differential power analysis (DPA). DPA collects a number of traces instead of one and calculates the average. The average power consumption is assumed to be different for different key values. For instance, this technique can be used to leak information from Data encryption standard (DES). In DES, one bit of the final round left intermediate depends on an unknown subkey, with other known bits that can be observed from the ciphertext. The value of this bit and the contributing subkey are obtainable with power analysis, as it is inputted to an XOR gate.

The need to improve upon DPA is based on the premise that averaging does not enable the complete usage of information embedded in a power trace. The resulting next step of power analysis introduces the first hints of machine learning: the Template attack (TA) [47]. Labeled as "the strongest form of a side-channel attack in an information theoretical sense", TA introduces profiling (i.e., recording many traces) to model the noise under the effect of subkeys by multivariate Gaussian variables, rather than eliminating it by averaging. The end product model enables the calculation of the probability of subkey assumptions, where the highest one is selected. Detailed steps of the attack are as follows [85]:

- A noise-induced power measurement for each of the K situations (or subkeys) is assumed to be a multivariate Gaussian model.
- I points of interest are selected in power traces (for example, with the sum of differences method) - that will constitute the I independent variables in the model.
- Multivariate **probability density function (PDF)** is constructed by calculating K mean vectors and K covariance matrices using the I points.
- When testing a new trace, the probability of each I is calculated by using each of the K pdfs. The probabilities that belong to the same K are then combined/added.
- K that corresponds to the highest value is selected as the probable situation - for instance, the subkey being equal to a value.

Introduction of the effective TA provided a reference for power side-channel analysis. After this point, the proceeding studies always provided comparisons with TA, where it is not trivial to significantly improve upon. The hopes for improvement shifted to using other machine learning algorithms that do not rely on the bounding Gaussian variables for modeling. The first studies in this respect used SVM and MLP.

Power analysis with machine learning. Hospodar et al. (2011) used SVM as a first example of machine learning-based power analysis [48]. Their aim was to determine whether an S-Box output of the Advanced encryption standard (AES) [86] has a Hamming weight that is odd/even, greater than four; or the fourth bit is zero or one, using S-Box power measurements. To this end, first, they reduced the number of features from the power trace to two by using three-dimensionality reduction methods, such as PCA. They consequently trained an SVM on a collection of traces using the reduced features. The results neither provide a significant improvement over TA, nor a full AES key recovery. Moreover, the authors did not elaborate on the dimensionality reduction aspect. For example, it is not clear why the authors used only two features.

Another such example that used MLP for power side-channel analysis is provided by Martinasek and Zeman (2013) [49]. In addition to S-Box, this study also included the add round key operation in power measurements in order to recover a subkey. Accordingly in the training phase, they averaged collected power traces for a subkey (for all possible subkeys) and provided them to a three-layered perceptron. To test a new power trace, they used the trained MLP. The highest activation in the output layer provides the subkey prediction of the neural network. Like the previous case, the results of this study did not provide a significant improvement over TA. However this time, a machine learning algorithm managed to map power traces to whole subkeys.

A final example of using ANNs for DPA is provided by Gilmore et al. (2015) [50]. Here, the authors target masked (protected) AES implementations. On collected traces, they first use PCA to reduce data dimension and then use an ANN to select 50 features. They show that key recovery from these features is equivalent to key recovery from unprotected AES.

To improve the performance, researchers started to look into different machine learning algorithms. In the 2010s, there was a resurgence of deep learning. Especially CNNs gained importance, with their remarkable success with image processing. Accordingly, power side-channel analysis researchers started to use CNNs, instead of earlier examples of SVMs and MLPs. One of the earliest such studies was provided by Maghrebi et al. (2016) [5]. In this study, the aim was to obtain several S-Box outputs from the first round of AES algorithms with different levels of protection. The comparison with TA reveals that the performance increased with using deep learning for profiling, where CNN is observed to have a good performance overall. Cagli et al. (2017), in a later study, paired data augmentation techniques with CNN to improve profiling [6]. Data augmentation is used to increase the amount of training data by using the already available data. In this study, the authors generated new data by adding distorting effects (like random delays and clock jitter) on top of traces. The experiments against AES implementations with software and hardware countermeasures showed that the resulting CNN is robust and successful. In a study by Kwon et al. (2020), the authors used an autoencoder to eliminate the noise and hinder the obfuscation effects from the collected power traces. When this was used as a preprocessing technique, it was seen that the power analysis effectiveness was improved [51]. Maghrebi (2020) investigated the performances of MLP and LSTM in addition to a CNN for DPA attack against AES. His results on the DPA Contest V2 dataset [87] show that while all three machine learning methods are effective in leaking the key, CNN outperforms the other two [52]. Das et al. (2019) successfully performed cross-device power analysis (profiling and attack traces are obtained from different devices) using a deep MLP [83]. Finally, Maghrebi (2019) investigated the practicality of deep learning-based power analysis under different realistic conditions [88].

Today, state of the art in power side-channel analysis is to use CNNs in profiling, where many researchers are attracted to use this method. This actuality is the focus of investigation in a study by Picek et al. (2018) [7]. In this study, the authors questioned the merits of using CNNs in the context of power analysis. In the case of DES, CNN performs well compared to other machine learning algorithms; especially in some specific conditions regarding feature size, dataset size, protection level, and so on. However, simpler techniques such as a random forest could perform similarly or even better when the dataset size is small or data contains a high level of noise. As the CNN performance is not consistently better, always using a machine learning algorithm for visual tasks in analyzing power measurements is questionable.

As a final angle, a study by Timon (2019) shows that it is even possible to use a deep MLP or CNN architecture (for desynchronized traces) to leak key information for the non-profiled attack (i.e., there is a limited number of power traces available to the attacker). They achieve this by training on the limited number of trace-key guess pairs and selecting the key that results in the best training

performance [53]. In a similar manner, Mukhtar et al. (2020) also investigated the performance of CNN with a limited number of traces of cryptosystems with SCA countermeasures. This study shows that while using more data and features benefits the classification, it is certainly possible to use limited data in the attack [54].

4.1.2 Sound Analysis. Sound analysis is another type of side-channel analysis where machine learning is commonly employed. The aim of the sound analysis is to reconstruct a secret from the emitted sound of an I/O device. Examples are provided by Asonov and Agrawal (2004), Zhuang et al. (2009), and Backes et al. (2010) [55–57]. The studies by (Asonov and Agrawal (2004) and Zhuang et al. (2009) dealt with sounds emitted from keyboards to construct typed letters, and the study by Backes et al. (2010) dealt with printer sounds to reconstruct printed documents. All these studies follow the same three-step procedure of feature extraction, training, and evaluation; which are described below.

- *Feature extraction.* All these studies extracted their features from the frequency domain, where keyboard-based studies used individual letters, and the printer-based study used whole words. For obtaining these features; Asonov and Agrawal (2004) used peak frequencies, Zhuang et al. (2009) used cepstrum bands, and Backes et al. (2010) used sub-bands. Cepstrum bands are a variation of the sub-bands technique, as cepstrum bands have a different way of aggregating the information in a particular sub-band [89].
- *Training.* In the training phase, (Asonov and Agrawal (2004) used an MLP, where the previously obtained letter sound features are used for training by backpropagation. The case of Zhuang et al. (2009) is more complex, as the authors used both unsupervised and supervised recognition. In the unsupervised part, they first performed K-means clustering, which is followed by an HMM. In clustering, they separated different letters into different clusters without any labeling. The conversion of clusters into letter labels is then achieved by the HMM. The supervised part used either a neural network, Linear discriminant analysis (LDA), or a Gaussian mixture; where labels determined by the unsupervised part are supplied alongside the letter features. This supervised part is intended to improve the unsupervised part. Lastly, Backes et al. (2010) constructed a dictionary from the word sound features. To increase the accuracy, an HMM is additionally constructed on triple word frequencies. This HMM is used to link the gap between the decision coming from the dictionary and how likely it is in the confines of the language.
- *Evaluation.* In the evaluation phase, these studies use their methods to obtain secrets. (Asonov and Agrawal (2004) extended their success in identifying keyboard letters to identifying ATM and telephone pads. Furthering this, Zhuang et al. (2009) showed that they can even reconstruct texts. They also reported that their method works better in identifying meaningful words, rather than random letters. Finally, Backes et al. (2010) attained high document reconstruction accuracy when they constructed their HMM using a domain-related corpus, rather than a general one. However, their applicability is very limited. Their method only works when using a specific (and older) type of printer, and the recording device that records the printer sounds should be very close to the printer.

To the knowledge of the authors, there are no further improvements in this area of sound side-channel analysis. This is particularly interesting, since there have been many improvements in speech processing and recognition. In other domains, many researchers use RNNs for this task. Furthermore, a specific RNN architecture called LSTM is shown to be very successful in recognizing speech with a large vocabulary [90]. Therefore, sound side-channel analysis can be revisited for much further improvements, especially for the equipment used today.

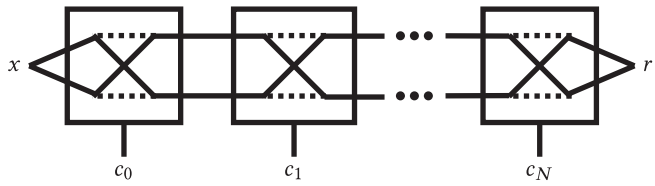

Fig. 11. Functional representation of a delay based (strong) arbiter PUF [91].

Table 5. Summary of Machine Learning-based Attacks on PUFs Studies

Study	Target PUF	Aim	ML tool
[58]	normal/XOR/feed-forward arbiter, lightweight secure, RO	challenge-response modelling	logistic regression, evaluation strategies
[59]	arbiter	challenge-response modelling	MLP, SVM
[60]	RO	challenge-response modelling	MLP, SVM, logistic regression

4.2 Attacks on PUFs

PUFs are security primitives that map a set of input (challenge) to a unique set of output (response). Process variations in each device enable this function [91]. PUFs can be grouped into two: weak and strong. The difference between them is the number of challenge-response pairs. The weak PUFs feature a very limited number of pairs, thus they must be hidden from outside access [58]. Strong PUFs on the other hand, feature a lot of publicly available pairs.

To understand why strong PUFs can disclose their challenge-response pairs, consider the strong arbiter PUF in Figure 11. In this type of PUF, the challenges (indicated by c_is in the figure) determine the path that the signals will take. The signal, x, is divided into two parts and enters a race, upon the paths determined by the challenges. The first arriving signal part will determine the response r to be zero or one. Given a challenge, a PUF will always give the same response. While this unpredictability made it hard to model the challenge-response behavior of arbiter PUFs before, people soon realized that using machine learning is an effective strategy. Such studies are summarized in Table 5 and discussed in detail afterward.

Rührmair et al. (2010), Hospodar et al. (2012), and (Kumar and Niamat (2018) provided important examples of such attacks [58–60]. The main difference between these studies is the PUFs that they attack. Rührmair et al. (2010) proposed mathematical models for multiple PUFs: arbiter, XOR arbiter, feed-forward arbiter, lightweight secure, and Ring oscillator (RO). For instance, they proposed a linear delay model for the arbiter PUF. On the other hand, Hospodar et al. (2012) worked on an actual CMOS arbiter PUF, and Kumar and Niamat (2018) targeted widely used RO-PUF. Their three-stage machine learning procedures are quite similar, as described below.

- *Feature extraction.* In all studies, the features are the challenges and the labels are their responses. Naturally, Rührmair et al. (2010) obtained these from their mathematical models, while Hospodar et al. (2012) obtained these from the Integrated circuit (IC)s. As Kumar and Niamat (2018) also used an FPGA implementation of RO-PUFs, their challenge-response data also comes from real hardware.
- *Training.* Rührmair et al. (2010) used logistic regression and evolution strategies (this method can be considered similar to genetic algorithms). To train logistic regression, they used challenge-response pairs. For evolution strategies, they created random delay vectors and evaluated them on how well they captured the challenge-response behavior. The vectors

Table 6. Summary of Machine Learning-based SCA Prevention Studies

Study	Type	Prevented Attacks	Aim	ML tool
[61]	active (1)	Flush+Reload	preventing cache leakage	Gaussian process, MLP
[62]	active (1)	Flush+Reload, Flush+Flush	preventing cache leakage	logistic regression, SVM, LDA
[63]	active (1)	Flush+Reload, Flush+Flush, Prime+Probe	preventing cache leakage	naïve Bayes, MLP
[99]	active (1)	Flush+Reload, Prime+Probe	preventing cache leakage	MLP
[100]	active (1)	Flush+Reload, Flush+Flush, Prime+Probe	preventing cache leakage	MLP
[101]	active (1)	Flush+Reload, Flush+Flush, Prime+Probe	preventing cache leakage	Gaussian process
[64]	passive (2)	DPA	preventing key leakage	MLP

that succeed are evolved. Hospodar et al. (2012) used MLP and SVM; whereas Kumar and Niamat (2018) used logistic regression, SVM, and MLP.

- *Evaluation.* All studies successfully use their machine learning models to predict a sufficiently correct number of responses from unseen challenges, thus breaking the security of PUFs. Additionally, both Hospodar et al. (2012) and Kumar and Niamat (2018) found out that MLP generally performs better in attacking compared to other machine learning methods they tested.

The effectiveness of these machine learning algorithms nearly made the strong PUFs useless [92]. This has forced the researchers to seek other solutions in accomplishing PUF-based authentication, such as using weak PUFs instead [93–96].

Interestingly, researchers have found another way to design PUF-like devices lately: by using machine learning algorithms. Sankhe et al. (2019) used a CNN to learn the fingerprints of radio devices, which is then used to identify each one [97]. Chatterjee et al. (2018) likewise used a simple three-layer ANN to authenticate radio systems through their wireless signals [98].

5 MACHINE LEARNING BASED HARDWARE COUNTERMEASURES

In the following subsections, machine learning-based hardware countermeasures are examined. First, Section 5.1 presents SCA prevention. Second, Section 5.2 presents HT detection. Thereafter, Section 5.3 presents fault injection detection. Finally, Section 5.4 presents JTAG-based attack detection.

5.1 Side-channel analysis Prevention

In the literature, there are two ways to prevent SCA using machine learning: active and passive. These studies are summarized in Table 6 and further explained in their sections.

5.1.1 Active SCA Prevention. Active prevention includes the usage of detectors. Chiappetta et al. (2016), Mushtaq et al. (2018), and Wang et al. (2021) provided such studies [61–63]. All are based on detecting cache-based attacks. A cache-based SCA is executed in either of the two styles: trace or time-driven. An attacker observes cache hit/miss activity to conduct a trace-driven attack or observes the execution time (thus the number of cache misses) to conduct a time-driven attack [102]. All - Chiappetta et al. (2016), Mushtaq et al. (2018), and Wang et al. (2021) - aimed to detect

time-driven attacks. Namely, Chiappetta et al. (2016) aimed to detect an attack called Flush+Reload, Mushtaq et al. (2018) aimed to detect Flush+Flush in addition, and Wang et al. (2021) aimed to detect Prime+Probe further in addition (similar/related work by Wang are also provided in [99–101]).

Flush+Reload flushes data in particular memory locations. Then, the attack accesses the data from the same locations and measures the access time. If the access time is relatively small, it means that another process (victim) accessed this data earlier [103]. A stealthier Flush+Flush does not access the memory directly. It measures the timing difference when flushing a particular memory element, which gives information on whether that element was cached or not [104]. In contrast, Prime+Probe does not flush but rather fills certain caches and then waits for the victim process to operate. Afterward, the attack accesses the same locations and measures the time differences in retaining them. If some lines take more time to load, it means that the victim process has used these caches earlier [105].

To detect these attacks, all studies use the following similar three-step procedure, which we describe below.

- *Feature extraction.* To detect *spy* processes that attempt the aforementioned attacks, all studies proposed using data from hardware counters. For Chiappetta et al. (2016), the data consists of cache accesses per process. For Mushtaq et al. (2018), it additionally consists of cache misses, total number of CPU cycles, and number of branch mispredictions per process. Wang et al. (2021) used 16 values such L1/L2/L3 hits and misses during operation.
- *Training.* To determine whether a process is "spy" or not, Chiappetta et al. (2016) used Gaussian distribution and MLP. Mushtaq et al. (2018) used three machine learning algorithms for the same task: logistic regression, SVM, and LDA - which is originally a dimensionality reduction method. Wang et al. (2021) used naïve Bayes, in addition to an MLP.
- *Evaluation.* The results obtained by Chiappetta et al. (2016) shows that MLP adequately detects spy processes that use Flush+Reload, while outperforming the Gaussian distribution. However, the Gaussian distribution approach produced results faster. In the case of Mushtaq et al. (2018), LDA outperformed the other two machine learning algorithms and also was able to adequately detect both Flush+Reload and Flush+Flush for AES and Rivest–Shamir–Adleman cryptosystem (RSA) processes. Results of Wang et al. (2021) show a trade-off between naïve Bayes and MLP: the latter achieved a better attack detection rate, while the former achieved less latency.

All these studies illustrated the effectiveness of using machine learning methods as online classifiers in software. A work preceding Wang et al. (2021) from the same authors also showed that a similar technique can be used to detect the dangerous Spectre attack that exploits leaking cache information by exploiting speculative branching [106, 107]. On the other hand, in all these studies a trade-off was made between using effective machine learning algorithms and the performance overhead that they create.

5.1.2 Passive SCA Prevention. Passive prevention against side-channel analysis is obtained by obfuscating the side-channel leakage, using machine learning. Aljuffri et al. (2020) proposed to replace the S-Box with a neural network in AES implementations. This breaks the linear correlation between the processed bits in S-Box and consumed power [64]. Their three-step procedure is as follows.

- *Feature extraction.* The feature extraction of this study is trivial, as they want to obtain the input-output behavior of an S-Box with a neural network. Thus, the features of the network are all 256 8-bit numbers, and their labels are their S-Box mappings.

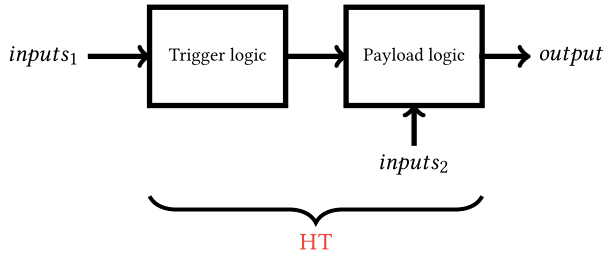

Fig. 12. A general representation of HTs [109].

Table 7. Summary of Machine Learning-based HT Detection Studies

Study	Type	Detection Medium	ML tool
[65]	reference-based (1)	power traces	SVM
[66]	reference-based (1)	netlist	SVM
[67]	reference-free (2)	power traces	PCA
[68]	reference-free (2)	power traces	PCA + SVM
[69]	reference-free (3)	circuit image	SVM
[70]	reference-free (3)	netlist	K-means
[71]	reference-free (4)	netlist	SVM
[72]	reference-free (4)	RTL design	CNN

- *Training.* To obtain this mapping, the authors trained an MLP. Then, they validated whether all inputs are mapped to the correct output.
- *Evaluation.* For evaluation, they employed their trained MLP in a software AES implementation. They used two common side-channel attacks of DPA and Correlation power analysis (CPA) [108]. Evaluation results show that they were able to protect a secure AES implementation against these attacks.

This study is important in terms of exploring different use areas of machine learning for hardware security. Moreover, this study highlights a completely different aspect of machine learning. Therefore, investigating the *nonlinearity* of neural networks for developing countermeasures on top of their data analysis capabilities can be very beneficial. On the other hand, this method incurs a high latency (75 times slower than plain AES) in software and potentially a high overhead in hardware. Thus, there is clearly a need for performance or cost improvement while investigating this aspect of neural networks.

5.2 Hardware Trojan Detection

HT is a malicious circuit modification, as illustrated in Figure 12. An entity in the manufacturing chain can insert an HT to an IC with the aim of monitoring, modifying, or disabling its operation. An HT can either be always active or wait for a specific set of inputs to activate. In the latter case, the trigger logic activates the HT once it receives those input(s). Once triggered, the payload logic starts working, which hinders the reliability and security of the IC.

We can investigate the techniques that protect against this malicious modification in two main groups: reference-based and reference-free HT detection. Reference-free methods on the other hand do not rely on such a set. Reference-free detection can be further grouped into side-channel fingerprinting, image & signal-based feature extraction, and graph-based feature extraction. The studies that fall into those four categories are summarized in Table 7.

5.2.1 Reference-based Detection. A reference-based method requires a set of identical ICs that are known to be HT-free. This set of *golden* ICs is used for feature extraction. In this survey, we

investigate two reference-based studies by Iwase et al. (2015) and Hasegawa et al. (2016) [65, 66]. Their three-step procedure of feature extraction, training, and evaluation are as follows:

- *Feature extraction.* To extract features, Iwase et al. (2015) first measured the power consumption of the devices. These traces were then transformed into the frequency domain. Resulting Discrete Fourier transform (DFT) values were directly used as features. Hasegawa et al. (2016) on the other hand, did not require a working IC to obtain features. They used the nets in a netlist to extract features such as fan-ins, number of flip-flop inputs and outputs, and so on.
- *Training.* Both studies used an SVM to differentiate between ICs that do and do not contain an HT. Additionally, Hasegawa et al. (2016) noted that the number of HT nets available to them was very scarce. This can be problematic during training; as SVM will prioritize to classify more numerous no HT cases correctly, at the expense of HT cases (this phenomenon is referred to as the *class imbalance* problem [110]). To counter this, they experimented with some data augmentation techniques.
- *Evaluation.* Iwase et al. (2015) reported their performance against 13 circuits that were implemented using FPGA, where 12 contain different HT types and one does not contain any HT. They successfully classified all. Hasegawa et al. (2016) reported their performance against individual nets. When they used an augmentation technique called *dynamic weighting*, which eliminates identical net instances and replicates HT nets to attain balance in amount, they were able to detect nearly all HT nets. However, they misclassified some normal nets as HT.

5.2.2 Side-channel Fingerprinting-based Detection. It is apparent that reference-based techniques are effective in detecting HTs. They obtain features from devices that are known to include HT or not and use supervised learning for effect. However, obtaining such a reference set is hard. Therefore, other methods are used to avoid this. One such method is referred to as side-channel fingerprinting [67, 68]. This method creates an IC model using side channels; such as power usage, temperature, electromagnetism, and path delay. Muehlberghuber et al. (2013) and Liu et al. (2014) proposed to use machine learning to statistically establish a side-channel fingerprint that ICs without HTs are expected to match [67, 68]. The three-step procedure of these studies is as follows.

- *Feature extraction.* Features used by Muehlberghuber et al. (2013) were power traces taken from actual chips that perform AES. These traces were then averaged for chips and the absolute difference traces from this mean are used as features. Liu et al. (2014) assumed that a golden/true Spice-level simulation model is available. Using this model, golden devices were simulated using the Monte Carlo method [111]. Simply explained, many devices were generated from a golden one by introducing random process variations. Simulated values of power consumption were used as side-channel features.
- *Training.* Both studies used PCA in classification. Muehlberghuber et al. (2013) used PCA on sample points selected from mean power traces, and reduced them to a single dimension. This enabled them then to construct a histogram, using all the power measurements of a chip. Liu et al. (2014) on the other hand, used PCA to reduce the dimension of power traces to train an SVM.
- *Evaluation.* Muehlberghuber et al. (2013) identified chips with HTs using the histogram of power measurement values. If a chip's values presented high variance, they concluded that it contains HT. By this method, they managed to group all chips successfully. The SVM used by Liu et al. (2014) however performed poorly in detecting chips with HTs. They consequently included Process control monitor (PCM) features (features obtained from the fabricated silicon) in their methodology, which increased the performance.

5.2.3 Image & signal-based Feature Extraction. Other methods to avoid a reference set include image processing [69] and extracting signal properties in a netlist [70]. Bao et al. (2014) and Salmani (2017) provided examples of these methods. The only difference between these studies is the way that they extracted their features. Their otherwise similar three-step procedure is as follows.

- *Feature extraction.* Image processing-based study by Bao et al. (2014) proposed to extract features during the reverse engineering after the imaging step (that is preceded by the steps of decapsulation and delayering). The images of the IC are consequently divided into grids. The content of these grids is compared to the grids of a golden image. The differences (such as area, the position of centroids, etc.) are taken as features. Salmani (2017) on the other hand, calculated the controllability and observability values for all signals in a netlist and used them as features.
- *Training.* The machine learning model used by Bao et al. (2014) is a one-class SVM. As there is no reference set, they made the assumption that most of the features were coming from HT-free instances. They used the SVM to put a boundary to encompass these training samples. Salmani (2017), by using unsupervised k-means clustering, also removed the need for a reference set. He divided the signals into three.
- *Evaluation.* Bao et al. (2014) classified an IC instance as normal if it fell inside their SVM boundary, and containing HT otherwise. For specific parameters (e.g., grid size) they obtained high accuracies in benchmark circuits with three different types of HTs. Salmani (2017) labeled the cluster with small features as genuine, and the cluster with the larger as HT related. This enabled him to achieve a perfect classification.

5.2.4 Graph-based Feature Extraction. A last line of research in using machine learning algorithms for detecting HTs without a reference set is to treat the design as a graph and extract features accordingly. Yu et al. (2020) treated the gate-level netlist as a directed graph [112], while Yasaei et al. (2021) treated the Register-transfer level (RTL)-level design as such [71, 72]. Their three-step procedure is as follows.

- *Feature extraction.* To extract features, first Yu et al. (2020) converted the gate-level netlist into a directed graph. Each node represents an instance (such as input, flip-flop, or various gates) and the directed edges indicate connections as well as the direction of signal flow. Then, they represented the sub-path information in this graph as features. Each of their features is a seven-element set (when the graph search depth is four) that contains node types in a path. Yasaei et al. (2021) on the other hand parsed the RTL design code to produce directed flow graphs, which connect each output signal to the input signal(s). Thereafter, they processed each node to generate vectors that contain related spatial information, which they used as features.
- *Training.* Yu et al. (2020) trained an SVM on their training set, by both using and not using class balancing - assigning more importance to the features that are taken from a HT infected circuit, as they are much smaller in number. Yasaei et al. (2021) used a CNN (also referred to as Graph neural network (GNN) as it acts upon a graph) to classify HT infected circuits, with also balancing both non-infected and infected classes.
- *Evaluation.* The results of Yu et al. (2020) show that class balancing creates a trade-off. When it was applied, more HT cases were detected but some uninfected cases were also classified as HTs, which is usually not acceptable. Yasaei et al. (2021) tested their CNN with benchmark circuits, such as ones implementing AES and RSA. Their detection results show that using a CNN for HT detection is a viable strategy, as it is not only able to work reference-free, but it is also able to detect unknown HTs.

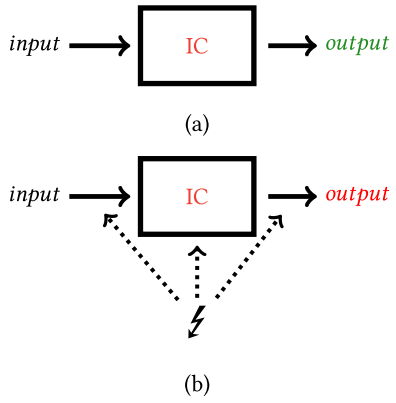

Fig. 13. (a) Normal, (b) faulty operation; where the aim of the attacker is to change the output of the device, by injecting faults during the operation.

Table 8. Summary of Machine Learning-based Fault Injection Detection Studies

Study	Device	Information Medium	ML tool
[73]	swarm bots	sensors	MLP
[74]	FPGA	sensors	naïve Bayes, LR, SVM, MLP
[75]	robot	sensors	decision tree
[76]	robot swarm	simulation	PCA
[77]	water level sensor	sensor	MLP
[78]	robots	sensors, commands, feedback	decision tree
[79]	water tanks	sensors	RNN
[80]	processor	instructions	RNN
[120]	processor	instructions	Hopfield network

While being mostly trivial, the machine learning usage in these studies illustrates the effectiveness of machine learning in many tasks. Furthermore, researchers relied on machine learning to make HT detection more practical, as they removed the reference sets. One point of underexploration however, is using Neural network (NN)s that are proven to be very successful in visual tasks. For example, there are very few works that explore CNNs for detecting HTs in circuit images [113].

5.3 Fault Injection Detection

Fault injection is the act of creating deliberate faults in hardware; in order to either affect the functionality of devices or steal data from them. This is illustrated in Figure 13. The means of fault injection includes (among others) voltage underfeeding [114], clock glitching [115], heating [116], using Electromagnetic (EM) waves [117], or lasers [118]. These techniques result in different effects on a device; ranging from data misreads, instruction skips, or misinterpretations to memory corruptions [119]. This can result in a faulty output, making the malicious goals of fault injection possible. The studies that use machine learning to detect injected faults are summarized in Table 8 and they are detailed below.

The first two studies use the sensor information to detect faults. Christensen et al. (2008) is based on hardware fault detection in autonomous robots through sensor data, where the faults are directly injected into the wheel structure of a swarm intelligence bot. They injected two types of faults: stuck-at-zero (corresponds to the abrupt stopping of a wheel) and stuck-at-constant (corresponds to the constant movement of a wheel at a particular speed) [73]. Shrivastwa et al. (2021) used digital sensors that are sensitive to environmental changes (such as clock/power variations and temperature) and injections such as laser and EM. They fed this data into an AI module, which decides if there is a fault attack on an FPGA board [74]. The three-step procedure of the two studies for detecting fault attacks is as follows.

- *Feature extraction.* Christensen et al. (2008) proposed to use various sensor data as features; the first type of sensor they obtain data from is infrared, which deals with light and proximity. This data is simple enough to be directly processed. The second data is from the camera, which provides a stream of colored images. These images were pooled (see CNNs in Section 2). That is, the image was divided into slices, and a slice was represented by a single number that indicates the distance to the closest object in the slice. Shrivastwa et al. (2021) used 16 timing sensors placed in an FPGA that runs AES. They recorded these sensor values as features during runs with and without faults, also noting the method of fault injection (clock glitching and EM).
- *Training.* Christensen et al. (2008) used a machine learning method referred to as *time-delay NN* [121]. This is a regular MLP, but a specific input contains current data as well as past data. As they dealt with swarm robots, which can work together in different environments to accomplish different tasks, they trained their network based on features for a particular situation. On the other hand, Shrivastwa et al. (2021) used two linear (Gaussian naïve Bayes, which assumes that the features follow a Gaussian distribution, affecting probabilities in the Bayes formula accordingly, and logistic regression) and two non-linear classifiers (SVM and MLP), which they trained with the extracted features belonging to both classes.
- *Evaluation.* The results of Christensen et al. (2008) show that they could effectively detect faults in a range of swarm configurations. Shrivastwa et al. (2021) found out that naïve Bayes performs the best in fault detection, which achieved >90% fault detection rate with 0 false alarms for most of the cases. Furthermore, this classifier is able to adequately differentiate between clock and EM attacks.

The importance of these studies is that they accomplish online fault detection (while the devices are in use). Similar studies include Stavrou et al. (2014) that detects faults in robots using a decision tree on sensor readings [75], Khaldi et al. (2017) that used PCA to detect faults in a simulated robot swarm [76], and Jäger et al. (2014) that used a time-delay MLP to detect faults in sensors [77]. Here, the usage of a time-delay NN as an alternative to RNNs is also an interesting choice, as it can reduce the latency introduced by the latter. Khalastchi et al. (2017) used unsupervised pattern detection in robotics to label a large collection of the sensor, measurement, and actuator feedback datasets as faulty or not. They trained a (supervised) decision tree based on this labeling. Their results show that this is an effective approach [78].

The second line of studies used RNNs to detect faults instead. The first such study is Przystalka (2008), where the author used an RNN to detect faults occurring in the water-level control system of multiple water tanks [79]. The second study Köylü et al. (2020) is ours, where we used a hardware RNN to detect faults in processor instructions of software RSA [80]. The three-step procedure of these studies is as follows.

- *Feature extraction.* Przystalka (2008) used sensor readings coming from water tanks, such as water levels, thresholds, and the like. In total, they use 10 such values at a time. They

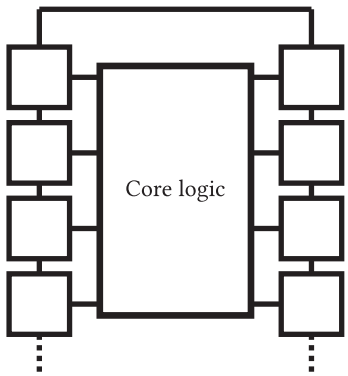

Fig. 14. Simplified JTAG boundary scan unit architecture.

have nine labels, where the first one is the normal conditions and the other eight are faults/problems: leakage from a tank, measurement errors, and so on. In Köylü et al. (2020), we extracted the features from non-faulty RSA decryptions. More specifically, we extracted the (11 control bits) of the instructions of these decryptions. Next, we grouped these instructions into a sequence of five and assigned the sixth instruction as the label.

- *Training.* Przystalka (2008) trained an RNN consisting of three layers, where only the second layer features feedback connections. The aim of the training is to make it detect not only the existence of a fault, but also the type of fault. In Köylü et al. (2020), we likewise trained an RNN with three layers, where the recurrent neurons are found in the second layer. This network learned to predict the sixth instruction when five previous are supplied. To elaborate further, after training, the RNN was able to produce a probability for all instructions that can follow the five previous ones. To establish a threshold to determine a faulty instruction, we calculated the lowest probability assigned to a correct instruction.
- *Evaluation.* Przystalka (2008) evaluated their network in two scenarios: (i) none or only one fault exists, and (ii) none or two faults exist at once. Their results show that RNN was able to improve the fault detection results, obtaining a good performance in both scenarios. In Köylü et al. (2020), we injected different faults into the instruction buffer to measure our detection performance. Any instruction that has a lower expectance probability was classified as a fault. Results show that we were able to detect effective faults that change the instructions to another in a near-perfect manner during the operation, while raising zero false alarms. However, detecting single-bit faults proved to be challenging.

There are a couple of important points here, especially arising from our study. First, we used a neural network as an online detector in hardware, showcasing such potential despite certain drawbacks (lower performance with bit faults and high hardware overhead compared to a RISC-V core). In Köylü et al. (2022), we further added the capability of correcting instruction faults by using a Hopfield network [120]. Second, we used a feedback architecture (an RNN), which is generally overlooked in favor of sequential architectures (e.g., MLP). Finally, this study also shows how using machine learning can provide flexibility, as we can protect multiple implementations by just loading different weights to the neural network before the operation.

5.4 JTAG-based Attack Detection

JTAG [122] boundary-scan architecture is developed to provide a scan path for ICs, which enables better testing. An example is illustrated in Figure 14. However, it also introduces a number of

Table 9. Summary of Machine Learning-based JTAG Attack Detection Studies

Study	Prevented Attack(s)	ML tool
[81]	reverse engineering	random forest, SVM

vulnerabilities in ICs. This is because JTAG enables access or control of some signals that are only intended for testing purposes [123]. To protect against potential attacks, a number of countermeasures were proposed, with each bringing their trade-offs. One such countermeasure is to disable the JTAG scan chain after manufacturing. However, this limits the in-field testability. Thus, using machine learning-based methods offers a new alternative to this limitation. One study that explored this is summarized in Table 9 and explained further below.

Ren et al. (2018) proposed using machine learning detectors in hardware to detect when the scan chain is being accessed for malicious purposes [81]. Their three-step procedure is as follows.

- *Feature extraction.* The authors proposed to use the scan chain instructions as features. They constructed two feature sets: the first one consists of instruction opcode, the number of clock cycles required, and whether the instruction and the transition are valid (as a flag); the second one consists of the opcodes of four recent instructions.
- *Training.* They used two different machine learning algorithms in this study: random forest and SVM. They also implemented a non-machine learning detector for comparison.
- *Evaluation.* The final evaluation is accomplished in hardware during operation. The implementation of these hardware machine learning detectors was made at the RTL level. The tree units that constitute the random forest are implemented in parallel and each has a memory, where their trained parameters are loaded from the main memory during power-up. The classification is made after a majority vote from the results of the tree units. In a similar manner, the SVM unit has also a memory to hold the trained parameters. For efficiency, a lookup table is used to implement the nonlinear RBF kernel. The evaluation results show that especially SVM provided a good detection of attacks. However, it introduced a large hardware overhead.

The results of this study are important in the future of hardware security and its relation to machine learning. First, it shows that machine learning can be effectively used as an online detector in hardware. This is typically not explored in other fields, especially in HT detection. Second, it also shows the main limitation of such an approach: the overhead. However, some of the same authors also presented a study where they reduced the number of features without significantly affecting the random forest classification accuracy of internal JTAG-based attacks [124].

6 TARGETING MACHINE LEARNING ALGORITHMS: ATTACKS AND COUNTERMEASURES AGAINST NEURAL NETWORKS

Machine learning algorithms, especially neural networks, are increasingly being used in safety-critical applications. One very relevant example is self-driving cars, where neural networks are used for both detecting objects on the road [125] and taking decisions [126, 127]. Thus, both attacks against the hardware that runs them and corresponding countermeasures are already being researched. The following subsections focus on these in the context of NNs.

6.1 Attacks Against NNs

Regardless of whether an NN is being run in software or as a hardware accelerator, a hardware attack against them has either of the two objectives: information leaking (e.g., stealing training

data, reverse engineering the architecture, obtaining weight values) or denial of correct service. These goals can be obtained by the following attack techniques.

Side-channel analysis. As in SCA in general, a number of techniques have been demonstrated to leak NN information. These include EM emanations, power readings, and using cache information. By leaking the information from these channels of a microprocessor, Batina et al. (2019) recovered the supplied inputs for an MLP [128]. On the other hand, Batina et al. (2018), Liu and Srivastava (2020), and Jeon et al. (2021) used side-channel information to obtain the parameters of MLPs and other networks: weight values, number of layers, neurons, and so on [129–131]. Likewise, Wei et al. (2018) and Yu et al. (2020) targeted ANN accelerators to obtain input and architecture/weight information of the NNs [132, 133].

It is apparent that this field of study of leaking NN information is in its infancy, i.e., there is a lot more area of exploration. Especially, realistic threat models are not established. This stems from the fact that most widely used NN architectures are available to everybody; such as AlexNet [134] and GoogleNet [135]. Likewise, leaking the input of an NN is a valid goal for a very limited number of cases when the input is sensitive: medical data, privacy-sensitive survey information, and others. However, in these cases, there are already solutions that enable users to provide the data to an NN in encrypted form [136]. Thus, simply leaking these encrypted inputs will not provide any benefits to an attacker. The studies in this field should either evolve to break encrypted inputs or motivate the applicable situations.

HT. Being invasive, HTs can be used for both information leaking and denial of a correct operation when they are placed in NN accelerators. Although not explored explicitly, HTs can be used to leak sensitive information about the NN, such as weight values [109].

There are however studies that explicitly illustrate how HTs can be used for misclassifications. (Clements and Lao (2018) and Clements and Lao (2019) both inject HTs into NN accelerators. Their aim is the same, i.e., creating misclassifications with minimal hardware footprints. When triggered, they perturb the operation by perturbing the neuron calculations [137] or the ReLU activation function calculation [138].

The previous two studies work very similarly to fault injection when the HT is activated. However, being integrated with the accelerator, they bring more possibilities. Ye et al. (2018) and Liu et al. (2020) use this property, where they completely change the output when triggered via certain inputs [139, 140]. This line of studies is more effective to realize the full power of putting malicious circuits in an NN accelerator.

Fault injection. Similar to HTs, fault injection enables an attacker to realize both attack goals (information leak and denial of correct operation). Considering the first goal, Breier et al. [141] used bit-flips to determine the parameter values of the last hidden layer of a network [141].

The second goal of denying a correct operation using fault injection is much more investigated. First, it is possible to inject faults into the input, making the NN decision faulty. While this is not explored explicitly, the concept of providing synthetic inputs that cannot be distinguished by eye but result in a very different decision exists (i.e., adversarial inputs) [142]. Second, a very common approach is to inject faults into the parameters: weights, biases, activation functions, and the like. Breier et al. (2018) used laser injections to a microcontroller and corrupted the calculation of activation functions (ReLU, softmax, Sigmoid, and tanh) [143]. Li et al. (2017) simulated fault injections (bit-flips) into the datapath and buffers [144]. Finally, Rakin et al. (2019) again used bit-flips, this time targetting the Dynamic random access memory (DRAM), where the weight values of the NN are typically stored [170]

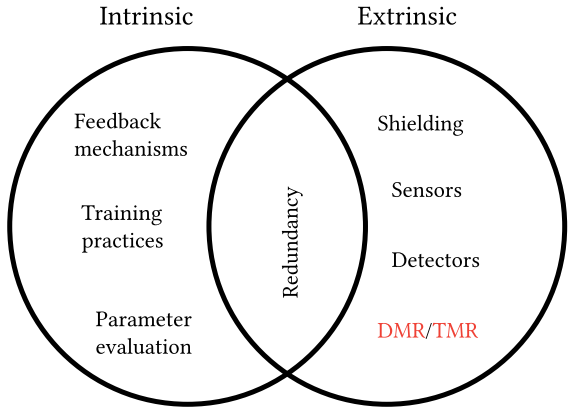

Fig. 15. Countermeasure categorization for protecting NNs from fault injection attacks.

This field of study, especially using fault attacks to deny the correct operation, is very relevant today. Causing misclassifications on an NN that is used for object detection in a self-driving car can easily cause accidents. Thus, this field will expand with the discovery of new attack techniques and hardware vulnerabilities that enable injecting faults remotely, such as the Rowhammer attack [145].

6.2 NN Countermeasures

Despite the long prominence of NNs, protecting them against hardware attacks did not receive the same attention. In the following, we will list the countermeasures against the attacks mentioned above. However, as countermeasures for NNs is not an established field, there are no explicit studies that link all attack techniques to a countermeasure. For these cases, we will still note if an existing countermeasure can be applied.

Side-channel analysis. To the authors' knowledge, there are no explicit studies that propose specific countermeasures to prevent SCA-based information leakage from neural networks. Theoretically, the widely investigated SCA countermeasures can also be used to protect NNs. These include obfuscation and balancing of side channels, such as the power consumption [146]. However, obfuscation and balancing are algorithm-specific techniques that are widely available for cryptosystem implementations, but not for NN implementations. Thus, there is a need for developing these protections for NNs, when the NN as an Intellectual property (IP) has to be protected.

HT. There are two possible ways to counter an HT inserted into the circuit of an NN accelerator: detection or countering the effect. In case the HT provides a backdoor for information leak, the only possible way is to use the already established HT prevention and detection techniques for digital circuits, with also using machine learning in various stages of detection [109].

To counter the effects (i.e., perturbations) of HTs on NNs, fault detectors or redundant mechanisms can be used. Further discussion on this is reserved for the fault injection part.

Fault injection. In contrast to other attack techniques, NN countermeasures against fault injection attacks are the most developed, partly because preventing fault attacks is synonymous with preventing faults. We can investigate them in two categories: intrinsic and extrinsic [147]. Figure 15 illustrate the groups, as well as redundancy, which can be applied both intrinsically and extrinsically.

The intrinsic countermeasures rely on the inherent fault-tolerant mechanisms of NNs. Some network architectures can tolerate faults better, such as Hopfield networks with their feedback mechanisms [148]. Some can be made more tolerant through various training practices. Ito and Takanami (1997) injected faults into some neuron outputs during each training iteration [149]. Another way is to use custom training algorithms for fault tolerance. Tan and Nanya (1994) proposed a learning algorithm that minimizes the number of essential links (i.e., weights which cause a significant change in the output if perturbed) through the objective function, thus making the trained network more tolerant to random faults [150]. Similarly, Mak et al. (2011) introduced two functions to measure training error, for faults that cause neuron and weight disconnections [151]. The learning algorithm proposed by Cavalieri and Mirabella (1999) on the other hand aims to balance the absolute weight values during training, in order to prevent faults in very large weight values affecting performance significantly [152]. Finally, Su et al. (2016) proposed to use the Genetic algorithm (GA) technique to make fault tolerance assessments between multiple training operations, using performance under faults (i.e., perturbations in parameters) [153]. A final example of using the training to generate a more fault-tolerant NN was presented by Xia et al. (2017), where they only write large enough weight updates to the Resistive random access memory (RRAM) to increase memory lifetime, as well as introduce remapping of stored values in case of faults, to make the NN implementation more tolerant against faults [171].

Another way of making NN implementations more resistant to faults is to evaluate their parameters post-training. Neti et al. (1992) introduced a formula to determine (and further to optimize) the fault tolerance of an NN given its weights [154]. Similarly, Sum et al. (2006) provided a function to evaluate the prediction accuracy under faults given the architecture, trained weights, and the training set [155]. Lastly, Reagen et al. (2018) provided a framework that injects fault to determine the negative impact on a Deep neural network (DNN) [156].

The final way of applying intrinsic countermeasures is to apply intrinsically redundant mechanisms, i.e., augmenting the NN architecture. Emmerson and Damper (1993) and Phatak and Koren (1995) replicated the last hidden layer neurons by making adjustments to preserve the correctness of the result [157, 158]. Similarly, Medler and Dawson (1994) replicated the structure of hidden-to-output layers, which are then connected to final output units [159]. All these methods achieve the same goal: increasing the attack surface so that faults do not cause misclassifications as easily. While these methods were proposed for basic MLP (before the deep learning era), they are still applicable to deeper DNN architectures, as most of them include standard/dense hidden and output layers. However, DNNs that are commonly used today have much more layers that precede those dense layers, where fault injections create significant problems [147]. This issue can be generalized to all intrinsic protections, which do not enable the protection of selected elements or the whole network [160].

This issue with intrinsic protections brought the necessity of applying extrinsic countermeasures, i.e., considering NNs as a part of a system and protecting accordingly. The first way to attain this is by shielding, i.e., meshes that cover the circuit from EM-based fault attacks. The second way to attain this is by using sensors, such as voltage and temperature to detect sudden changes [119]. The third way, which can be more specialized for protecting NNs is to use detectors. Wang et al. (2020) proposed to learn the layer-wise neuron activation rate when normal inputs are provided. Thereafter, they were able to differentiate adversarial/faulty inputs from the changes in the activation rates [142]. Our work, (Köylü et al. (2021)) expanded upon this idea by presenting two types of neuron activation rate-based detectors against faults that affect NN weights and biases. In this work, we proposed to check a reference input-output pair periodically to eliminate persistent faults, in addition to monitoring layer activation rates to detect temporary faults [147].

While the aforementioned methods are effective, the most effective extrinsic countermeasure is applying Dual modular redundancy (DMR) for fault detection [161] and Triple modular redundancy (TMR) for correction [162]. However, DMR or TMR for software NN applications create x2/x3 latency and x2/x3 overhead for hardware accelerators. Thus, applying selective redundancy by using inherent properties of NNs is an emerging field of study. Such a study was provided by Li et al. (2019), where they introduced a method to determine the importance of neurons to selectively protect against faults. Their method consists of assigning neuron importance from the absolute summation of the weights that leave it [163]. Venkataramani et al. (2014) and our work (Köylü et al. (2022)) proposed to use a more sophisticated backpropagation algorithm to calculate neuron importance instead, by using a subset of the training data [160, 164]. Here, Venkataramani et al. (2014) used the average updates over the subset; whereas in Köylü et al. (2022), we used absolute summation and variance. We further proposed selective granularity: the ability to protect individual weights/biases, neurons, or layers. In addition to this, Ruospo et al. (2022) proposed to prune unimportant neurons to minimize the added overhead [165]. Finally, Schorn et al. (2020) proposed to use the layer-wise propagation algorithm [166], which calculates the importance of a neuron from the importance of neurons that are connected to it in the proceeding layers. It determines the last layer neuron references from their outputs after an inference operation [167]. Thus, this method also requires a portion of the dataset to construct these values.

This field of selective redundancy using inherent NN properties is very promising: it aims to achieve the effectiveness of DMR/TMR more efficiently. However, all of the aforementioned studies use some kind of heuristics to determine important weights and neurons. Current approaches have already started to become incremental, rather than significant improvements. This improvement can be achieved by theoretically proposing methods that aim to achieve maximal fault tolerance.

7 CONCLUSION

This survey presented a literature cover on hardware security in the context of machine learning. To our knowledge, this is the most complete overview of this topic; as we covered all attack, countermeasure, and threat/protection of NN angles. Additionally, one of our main aims was to address what is possible, in addition to what exists. We hope that this survey guides hardware security engineers in using machine learning, and attracts AI engineers to hardware security as a potential field of operation.

In the remainder of this section, we first provide a summary of each hardware security subsection (Section 7.1). Then, we provide future directions for machine learning in hardware security (Section 7.2).

7.1 Summary

The following is a brief summary of each hardware security subsection.

- **Side-channel analysis:** SCA is probably the hardware security area with the most reliance on machine learning. Aside from a couple of sound-based studies that rely on outdated equipment (i.e., old keyboards and printers), this is mainly due to power SCA. The use of deeper ANNs (i.e., CNNs) has made it possible to break cryptosystem keys in various circumstances. The issue here is, this CNN usage heavily relies on external trends. There is a need to find more suitable algorithms that infer cryptographic keys from power traces.
- **Attacks on PUFs:** Use of machine learning had a drastic effect on the development of PUFs. Modeling of the challenge-response behavior using machine learning algorithms made it hard to rely on strong PUF-based security. Thus, weak PUFs and making them more reliable and secure gained much more importance.

- **Side-channel analysis prevention:** Most of the focus in this area is on detecting information leaks through caches. Use of machine learning algorithms on auxiliary information such as the number of cache accesses or CPU clock cycles helped to detect the most prevalent attacks. There is also a newly emerging area in SCA prevention: using neural networks for obfuscation.
- **Hardware Trojan detection:** Another area with a high level of reliance on machine learning is HT detection. Existing work demonstrates the effectiveness of machine learning algorithms on various types of features (e.g., image or netlist-based) in detecting these malicious circuits. Furthermore, machine learning also helps with cases when there is no reference set of trusted devices.
- **Fault injection detection:** Most of the earlier work in this domain was on robotics. These works used machine learning algorithms to determine when there are faults, by investigating data coming from various sensors. In the last years, we introduced another line of studies in this domain: using machine learning algorithms as hardware modules to detect faults in processor instructions.
- **JTAG-based attack detection:** There is only one study that used machine learning to detect JTAG-based attacks. Moreover, this study is one of the first instances to use a machine learning algorithm in hardware, as an online detector.
- **Attacks against NNs:** Except for some work about leaking ANN information like architecture and weight values, the most relevant threat in this domain is injecting faults to make the inference faulty. This can cause devastating results, such as accidents in the case of automated driving.
- **NN countermeasures:** Accordingly, most of NN countermeasures focus on protecting NNs from fault attacks. Earlier works investigated the inherent fault tolerance of NNs. Later work proposed external protection such as detectors, sensors, and applying DMR/TMR. Lately, using inherent NN properties to apply selective redundancy gained recognition, but still has unexplored potential.

7.2 Future Directions

In this section, we highlight re-occurring themes in the subsections summarized in the previous Section 7.1, in order to provide future directions.

The first common theme in machine learning usage for many hardware security domains is the reliance on external trends and missing other potential. The most prominent example is using visual task-oriented CNNs for processing time-series power traces in DPA, due to the recent fame of these networks. A natural next step in DPA however is to explore networks that can explain time-series data better, such as an RNN [168]. Another example is for HT detection. In this domain, very few examples explore DNNs that are very successful in visual tasks. Thus, hardware security engineers should consider exploring or developing machine learning algorithms that fit the task, rather than using what is popular or most available.

The second theme is only using machine learning as an *offline* data processing tool. As this survey illustrates, machine learning is indeed very effective to be used as such. However, domains like automated driving show us that there is great potential in using machine learning algorithms as *online* decision tools. We believe that cache attack prevention methods and our work on detecting instruction faults [80] illustrated this potential. However, an essential challenge in online usage is the latency and area overheads these algorithms incur.

The third and final theme is not updating previous ideas. Studies have illustrated the successful usage of machine learning in decoding text from keyboard and printer sounds. However, these are for outdated hardware and there are no novel studies for newer equipment. This is a missed

opportunity because of the following: (i) machine learning algorithms had a very significant performance increase since, and (ii) there are many more devices leaking information in the current Internet of things (IoT)-era

REFERENCES

[1] Peter W. Singer and Allan Friedman. 2014. *Cybersecurity: What Everyone Needs to Know*. Oxford University Press.

[2] Chee-Wooi Ten, Chen-Ching Liu, and Govindarasu Manimaran. 2008. Vulnerability assessment of cybersecurity for SCADA systems. *IEEE Transactions on Power Systems* 23, 4 (2008), 1836–1846.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484.

[4] Yaniv Leviathan and Yossi Matias. 2018. Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone. (May 2018). https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html.

[5] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 3–26.

[6] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 45–68.

[7] Stjepan Picek, Ioannis Petros Samiotis, Annelie Heuser, Jaehun Kim, Shivam Bhasin, and Axel Legay. 2018. On the performance of deep learning for side-channel analysis. *International Association for Cryptologic Research* (2018).

[8] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. 2020. Applications of machine learning techniques in side-channel attacks: A survey. *Journal of Cryptographic Engineering* 10, 2 (2020), 135–162.

[9] Roel Maes and Ingrid Verbauwhede. 2010. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*. Springer, 3–37.

[10] Ulrich Rührmair and Jan Sölter. 2014. PUF modeling attacks: An introduction and overview. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[11] M. Tanjidur Rahman, Qihang Shi, Shahin Tajik, Haoting Shen, Damon L. Woodard, Mark Tehranipoor, and Navid Asadizanjani. 2018. Physical inspection & attacks: New frontier in hardware security. In *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. IEEE, 93–102.

[12] Zhao Huang, Quan Wang, Yin Chen, and Xiaohong Jiang. 2020. A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access* 8 (2020), 10796–10826.

[13] Shamik Kundu, Xingyu Meng, and Kanad Basu. 2021. Application of machine learning in hardware trojan detection. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 414–419.

[14] Rana Elnaggar and Krishnendu Chakrabarty. 2018. Machine learning for hardware security: Opportunities and risks. *Journal of Electronic Testing* 34, 2 (2018), 183–201.

[15] Wenye Liu, Chip-Hong Chang, Xueyang Wang, Chen Liu, Jason Fung, Mohammad Ebrahimabadi, Naghmeh Karimi, Xingyu Meng, and Kanad Basu. 2021. Two sides of the same coin: Boons and banes of machine learning in hardware security. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).

[16] Tom M. Mitchell et al. 1997. *Machine Learning. WCB*. McGraw-Hill Boston, MA.

[17] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin.

[18] Michael I. Jordan and David E. Rumelhart. 1992. Forward models: Supervised learning with a distal teacher. *Cognitive Science* 16, 3 (1992), 307–354.

[19] T. Cover. 1968. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory* 14, 1 (1968), 50–55.

[20] J. Ross Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (1986), 81–106.

[21] Irina Rish et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, Vol. 3. IBM New York, 41–46.

[22] Scott C. Markley and David J. Miller. 2010. Joint parsimonious modeling and model order selection for multivariate Gaussian mixtures. *IEEE Journal of Selected Topics in Signal Processing* 4, 3 (2010), 548–559.

[23] Johan A. K. Suykens and Joos Vandewalle. 1999. Least squares support vector machine classifiers. *Neural Processing Letters* 9, 3 (1999), 293–300.

[24] Chih-Wei Hsu and Chih-Jen Lin. 2002. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks* 13, 2 (2002), 415–425.

[25] Lawrence R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

[26] G. David Forney. 1973. The Viterbi algorithm. *Proc. IEEE* 61, 3 (1973), 268–278.
[27] Warren S. McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5, 4 (1943), 115–133.
[28] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive Modeling* 5, 3 (1988), 1.
[29] Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14, 2 (1990), 179–211.
[30] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[31] John J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *PNAS* 79, 8 (1982), 2554–2558.
[32] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. 2020. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217* (2020).
[33] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. 2002. Image processing with neural networks-a review. *Pattern Recognition* 35, 10 (2002), 2279–2301.
[34] Yonghao Xu, Weikang Yu, Pedram Ghamisi, Michael Kopp, and Sepp Hochreiter. 2022. Txt2Img-MHN: Remote sensing image generation from text using modern Hopfield networks. *arXiv preprint arXiv:2208.04441* (2022).
[35] CS231n: Convolutional Neural Networks for Visual Recognition. ([n. d.]). http://cs231n.stanford.edu/.
[36] Hong Hui Tan and King Hann Lim. 2019. Vanishing gradient mitigation with deep learning neural network optimization. In *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. IEEE, 1–4.
[37] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. 2012. *Introduction to Linear Regression Analysis*. Vol. 821. John Wiley & Sons.
[38] Hyeoun Park. 2013. An introduction to logistic regression: From basic concepts to interpretation with particular attention to nursing domain. *Journal of Korean Academy of Nursing* 43, 2 (2013), 154–164.
[39] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. Unsupervised learning. In *The Elements of Statistical Learning*. Springer, 485–585.
[40] John A. Hartigan and Manchek A. Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.
[41] David J. C. MacKay et al. 1998. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences* 168 (1998), 133–166.
[42] Mark Richardson. 2009. Principal component analysis. *URL: http://people.maths.ox.ac.uk/richardsonm/SignalProcPCA.pdf (last access: 3.5. 2013). Aleš Hladnik Dr., Ass. Prof., Chair of Information and Graphic Arts Technology, Faculty of Natural Sciences and Engineering, University of Ljubljana, Slovenia ales. hladnik@ ntf. uni-lj. si* 6 (2009), 16.
[43] Randy L. Haupt and Sue Ellen Haupt. 2005. *Practical Genetic Algorithms*. Wiley.
[44] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*. 1096–1103.
[45] Geoffrey E. Hinton. 2009. Deep belief networks. *Scholarpedia* 4, 5 (2009), 5947.
[46] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.
[47] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2002. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 13–28.
[48] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. 2011. Machine learning in side-channel analysis: A first study. *Journal of Cryptographic Engineering* 1, 4 (2011), 293.
[49] Zdenek Martinasek and Vaclav Zeman. 2013. Innovative method of the power analysis. *Radioengineering* 22, 2 (2013), 586–594.
[50] Richard Gilmore, Neil Hanley, and Maire O'Neill. 2015. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 106–111.
[51] Donggeun Kwon, HeeSeok Kim, and Seokhie Hong. 2020. Improving non-profiled side-channel attacks using autoencoder based preprocessing. *Cryptology ePrint Archive* (2020).
[52] Houssem Maghrebi. 2020. Deep learning based side-channel attack: A new profiling methodology based on multi-label classification. *Cryptology ePrint Archive* (2020).
[53] Benjamin Timon. 2019. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019), 107–131.
[54] Naila Mukhtar, Louiza Papachristodoulou, Apostolos P. Fournaris, Lejla Batina, and Yinan Kong. 2020. Machine-learning assisted side-channel attacks on RNS-based elliptic curve implementations using hybrid feature engineering. *Cryptology ePrint Archive* (2020).

[55] Dmitri Asonov and Rakesh Agrawal. 2004. Keyboard acoustic emanations. IEEE, 3.
[56] Li Zhuang, Feng Zhou, and J. Doug Tygar. 2009. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security (TISSEC)* 13, 1 (2009), 3.
[57] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. 2010. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*. 307–322.
[58] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*. ACM, 237–249.
[59] Gabriel Hospodar, Roel Maes, and Ingrid Verbauwhede. 2012. Machine learning attacks on 65nm arbiter PUFs: Accurate modeling poses strict bounds on usability. In *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*. IEEE, 37–42.
[60] Sharad Kumar and Mohammed Niamat. 2018. Machine learning based modeling attacks on a configurable PUF. In *NAECON 2018-IEEE National Aerospace and Electronics Conference*. IEEE, 169–173.
[61] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing* 49 (2016), 1162–1174.
[62] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, and Guy Gogniat. 2018. NIGHTs-WATCH: A cache-based side-channel intrusion detector using hardware performance counters. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 1.
[63] Han Wang, Soheil Salehi, Hossein Sayadi, Avesta Sasan, Tinoosh Mohsenin, P. D. Sai Manoj, Setareh Rafatirad, and Houman Homayoun. 2021. Evaluation of machine learning-based detection against side-channel attacks on autonomous vehicle. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 1–4.
[64] Abdullah Aljuffri, Pradeep Venkatachalam, Cezar Reinbrecht, Said Hamdioui, and Mottaqiallah Taouil. 2020. S-NET: A confusion based countermeasure against power attacks for SBOX. In *International Conference on Embedded Computer Systems*. Springer, 295–307.
[65] Takato Iwase, Yusuke Nozaki, Masaya Yoshikawa, and Takeshi Kumaki. 2015. Detection technique for hardware trojans using machine learning in frequency domain. In *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*. IEEE, 185–186.
[66] Kento Hasegawa, Masaru Oya, Masao Yanagisawa, and Nozomu Togawa. 2016. Hardware trojans classification for gate-level netlists based on machine learning. In *On-Line Testing and Robust System Design (IOLTS), 2016 IEEE 22nd International Symposium on*. IEEE, 203–206.
[67] Michael Muehlberghuber, Frank K. Gürkaynak, Thomas Korak, Philipp Dunst, and Michael Hutter. 2013. Red team vs. blue team hardware trojan analysis: Detection of a hardware trojan on an actual ASIC. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 1.
[68] 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*. ACM, 1–6.
[69] Chongxi Bao, Domenic Forte, and Ankur Srivastava. 2014. On application of one-class SVM to reverse engineering-based hardware trojan detection. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on*. IEEE, 47–54.
[70] 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.
[71] Shichao Yu, Chongyan Gu, Weiqiang Liu, and Maire O'Neill. 2020. A novel feature extraction strategy for hardware trojan detection. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[72] 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)*. IEEE, 1504–1509.
[73] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, and Marco Dorigo. 2008. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots* 24, 1 (2008), 49–67.
[74] Ritu-Ranjan Shrivastwa, Sylvain Guilley, and Jean-Luc Danger. 2021. Multi-source fault injection detection using machine learning and sensor fusion. In *International Conference on Security and Privacy*. Springer, 93–107.
[75] Demetris Stavrou, Demetrios G. Eliades, Christos G. Panayiotou, and Marios M. Polycarpou. 2016. Fault detection for service mobile robots using model-based method. *Autonomous Robots* 40, 2 (2016), 383–394.
[76] Belkacem Khaldi, Fouzi Harrou, Foudil Cherif, and Ying Sun. 2017. Monitoring a robot swarm using a data-driven fault detection approach. *Robotics and Autonomous Systems* 97 (2017), 193–203.
[77] Georg Jäger, Sebastian Zug, Tino Brade, André Dietrich, Christoph Steup, Christian Moewes, and Ana-Maria Cretu. 2014. Assessing neural networks for sensor fault detection. In *2014 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE, 70–75.

[78] Eliahu Khalastchi, Meir Kalech, and Lior Rokach. 2017. A hybrid approach for improving unsupervised fault detection for robotic systems. *Expert Systems with Applications* 81 (2017), 372–383.
[79] Piotr Przystałka. 2008. Model-based fault detection and isolation using locally recurrent neural networks. In *International Conference on Artificial Intelligence and Soft Computing*. Springer, 123–134.
[80] Troya Çağıl Köylü, Cezar Rodolfo Wedig Reinbrecht, Said Hamdioui, and Mottaqiallah Taouil. 2020. RNN-based detection of fault attacks on RSA. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[81] Xuanle Ren, Francisco Pimentel Torres, R. D. Blanton, and Vítor Grade Tavares. 2018. IC protection against JTAG-based attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018).
[82] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. 2014. A primer on hardware security: Models, methods, and metrics. *Proc. IEEE* 102, 8 (2014), 1283–1295.
[83] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. 2019. X-DeepSCA: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.
[84] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.
[85] 2018. Template attacks. (May 2018). https://wiki.newae.com/Template_Attacks.
[86] Joan Daemen and Vincent Rijmen. 2013. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer Science & Business Media.
[87] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M. Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, et al. 2014. Practical improvements of side-channel attacks on AES: Feedback from the 2nd DPA contest. *Journal of Cryptographic Engineering* 4, 4 (2014), 259–274.
[88] Houssem Maghrebi. 2019. Deep learning based side channel attacks in practice. *Cryptology ePrint Archive* (2019).
[89] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. 2010. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint arXiv:1003.4083* (2010).
[90] Haşim Sak, Andrew Senior, and Françoise Beaufays. 2014. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128* (2014).
[91] G. Edward Suh and Srinivas Devadas. 2007. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 9–14.
[92] Arunkumar Vijayakumar, Vinay C. Patil, Charles B. Prado, and Sandip Kundu. 2016. Machine learning resistant strong PUF: Possible or a pipe dream?. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium*. IEEE, 19–24.
[93] Ulrich Rührmair and Daniel E. Holcomb. 2014. PUFs at a glance. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.
[94] Vinay C. Patil, Arunkumar Vijayakumar, Daniel E. Holcomb, and Sandip Kundu. 2017. Improving reliability of weak PUFs via circuit techniques to enhance mismatch. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 146–150.
[95] Md. Nazmul Islam, Vinay C. Patil, and Sandip Kundu. 2017. On enhancing reliability of weak PUFs via intelligent post-silicon accelerated aging. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 3 (2017), 960–969.
[96] Leandro Santiago, Vinay C. Patil, Charles B. Prado, Tiago A. O. Alves, Leandro A. J. Marzulo, Felipe M. G. França, and Sandip Kundu. 2017. Realizing strong PUF from weak PUF via neural computing. In *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 1–6.
[97] Kunal Sankhe, Mauro Belgiovine, Fan Zhou, Luca Angioloni, Frank Restuccia, Salvatore D'Oro, Tommaso Melodia, Stratis Ioannidis, and Kaushik Chowdhury. 2019. No radio left behind: Radio fingerprinting through deep learning of physical-layer hardware impairments. *IEEE Transactions on Cognitive Communications and Networking* 6, 1 (2019), 165–178.
[98] Baibhab Chatterjee, Debayan Das, Shovan Maity, and Shreyas Sen. 2018. RF-PUF: Enhancing IoT security through authentication of wireless nodes using in-situ machine learning. *IEEE Internet of Things Journal* 6, 1 (2018), 388–398.
[99] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2020. Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.
[100] Han Wang, Hossein Sayadi, Gaurav Kolhe, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2020. Phased-guard: Multi-phase machine learning framework for detection and identification of zero-day microarchitectural side-channel attacks. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 648–655.
[101] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2020. HybriDG: Hybrid dynamic time warping and Gaussian distribution model for detecting emerging zero-day microarchitectural side-channel attacks. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 604–611.

[102] Daniel Page. 2003. Defending Against Cache-based Side-channel Attacks. *Information Security Technical Report* 8, 1 (2003), 30–44.
[103] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*, Vol. 1. 22–25.
[104] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: A fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.
[105] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 605–622.
[106] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.
[107] Han Wang, Hossein Sayadi, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. 2020. SCARF: Detecting side-channel attacks at real-time using low-level hardware features. In *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–6.
[108] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 16–29.
[109] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. 2014. Hardware trojan attacks: Threat analysis and countermeasures. *Proc. IEEE* 102, 8 (2014), 1229–1247.
[110] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter* 6, 1 (2004), 20–29.
[111] Helmut G. Katzgraber. 2009. Introduction to Monte Carlo methods. *arXiv preprint arXiv:0905.1629* (2009).
[112] C. Vasudev. 2006. *Graph Theory with Applications*. New Age International.
[113] Richa Sharma, G. K. Sharma, and Manisha Pattanaik. 2021. A few shot learning based approach for hardware trojan detection using deep Siamese CNN. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. IEEE, 163–168.
[114] Nidhal Selmane, Sylvain Guilley, and Jean-Luc Danger. 2008. Practical setup time violation attacks on AES. In *2008 Seventh European Dependable Computing Conference*. IEEE, 91–96.
[115] Frederic Amiel, Christophe Clavier, and Michael Tunstall. 2006. Fault analysis of DPA-resistant algorithms. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*. Springer, 223–236.
[116] Sudhakar Govindavajhala and Andrew W. Appel. 2003. Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy*, Vol. 5.
[117] Jörn-Marc Schmidt and Michael Hutter. 2007. *Optical and EM Fault-attacks on CRT-based RSA: Concrete Results*.
[118] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria. 2010. How to flip a bit?. In *2010 IEEE 16th International On-Line Testing Symposium*. IEEE, 235–239.
[119] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. 2006. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE* 94, 2 (2006), 370–382.
[120] Troya Çağil Köylü, Moritz Fieback, Said Hamdioui, and Mottaqiallah Taouil. 2022. Using Hopfield networks to correct instruction faults. In *2022 IEEE 31st Asian Test Symposium (ATS)*. IEEE, 102–107.
[121] Daniel S. Clouse, C. Lee Giles, Bill G. Horne, and Garrison W. Cottrell. 1997. Time-delay neural networks: Representation and induction of finite-state machines. *IEEE Transactions on Neural Networks* 8, 5 (1997), 1065–1070.
[122] Colin Maunder. 1986. The Joint Test Action Group. *Computer-Aided Engineering Journal* 3, 4 (1986), 121–122.
[123] Kurt Rosenfeld and Ramesh Karri. 2010. Attacks and defenses for JTAG. *IEEE Design & Test of Computers* 27, 1 (2010).
[124] Xuanle Ren, R. D. Shawn Blanton, and Vítor Grade Tavares. 2018. Detection of IJTAG attacks using LDPC-based feature reduction and machine learning. In *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 1–6.
[125] Mehdi Masmoudi, Hakim Ghazzai, Mounir Frikha, and Yehia Massoud. 2019. Object detection learning techniques for autonomous vehicle applications. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE, 1–5.
[126] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).
[127] Mohammed Al-Qizwini, Iman Barjasteh, Hothaifa Al-Qassab, and Hayder Radha. 2017. Deep learning algorithm for autonomous driving using GoogLeNet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 89–96.
[128] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. Poster: Recovering the input of neural networks via single shot side-channel attacks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2657–2659.

[129] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2018. CSI neural network: Using side-channels to recover your artificial neural network information. *arXiv preprint arXiv:1810.09076* (2018).
[130] Yuntao Liu and Ankur Srivastava. 2020. GANRED: GAN-based reverse engineering of DNNs via cache side-channel. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*. 41–52.
[131] Hyeran Jeon, Nima Karimian, and Tamara Lehman. 2021. A new foe in GPUs: Power side-channel attacks on neural network. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 313–313.
[132] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. 2018. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 393–406.
[133] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 209–218.
[134] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.
[135] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.
[136] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. PMLR, 201–210.
[137] Joseph Clements and Yingjie Lao. 2018. Hardware trojan attacks on neural networks. *arXiv preprint arXiv:1806.05768* (2018).
[138] Joseph Clements and Yingjie Lao. 2019. Hardware trojan design on neural networks. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[139] Jing Ye, Yu Hu, and Xiaowei Li. 2018. Hardware trojan in FPGA CNN accelerator. In *2018 IEEE 27th Asian Test Symposium (ATS)*. IEEE, 68–73.
[140] Zizhen Liu, Jing Ye, Xing Hu, Huawei Li, Xiaowei Li, and Yu Hu. 2020. Sequence triggered hardware trojan in neural network accelerator. In *2020 IEEE 38th VLSI Test Symposium (VTS)*. IEEE, 1–6.
[141] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. 2020. SNIFF: Reverse engineering of neural networks with fault attacks. *arXiv preprint arXiv:2002.11021* (2020).
[142] Si Wang, Wenye Liu, and Chip-Hong Chang. 2020. Fired neuron rate based decision tree for detection of adversarial examples in DNNs. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[143] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2018. Practical fault attack on deep neural networks. In *SIGSAC*.
[144] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. 2017. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.
[145] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.
[146] Mottaqiallah Taouil, Abdullah Aljuffri, and Said Hamdioui. 2021. Power side channel attacks: Where are we standing?. In *2021 16th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 1–6.
[147] Troya Çağıl Köylü, Cezar Rodolfo Wedig Reinbrecht, Said Hamdioui, and Mottaqiallah Taouil. 2021. Deterministic and statistical strategies to protect ANNs against fault injection attacks. In *2021 18th International Conference on Privacy, Security and Trust (PST)*. IEEE, 1–10.
[148] George Bolt. 1991. Investigating fault tolerance in artificial neural networks. (1991).
[149] Takehiro Ito and Itsuo Takanami. 1997. On fault injection approaches for fault tolerance of feedforward neural networks. In *Proceedings Sixth Asian Test Symposium (ATS'97)*. IEEE, 88–93.
[150] Yasuo Tan and Takashi Nanya. 1994. A fault-tolerant multilayer neural network model and its properties. *Systems and Computers in Japan* 25, 2 (1994), 33–43.
[151] Shue Kwan Mak, Pui-Fai Sum, and Chi-Sing Leung. 2011. Regularizers for fault tolerant multilayer feedforward networks. *Neurocomputing* 74, 11 (2011), 2028–2040.
[152] Salvatore Cavalieri and Orazio Mirabella. 1999. A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks. *Neural Networks* 12, 1 (1999), 91–106.
[153] Feng Su, Peijiang Yuan, Yangzhen Wang, and Chen Zhang. 2016. The superior fault tolerance of artificial neural network training with a fault/noise injection-based genetic algorithm. *Protein & Cell* 7, 10 (2016), 735–748.

[154] Chalapathy Neti, Michael H. Schneider, and Eric D. Young. 1992. Maximally fault tolerant neural networks. *IEEE Transactions on Neural Networks* 3, 1 (1992), 14–23.
[155] John Sum, Chi-sing Leung, and Kevin Ho. 2006. Prediction error of a fault tolerant neural network. In *International Conference on Neural Information Processing*. Springer, 521–528.
[156] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. 2018. Ares: A framework for quantifying the resilience of deep neural networks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[157] Martin D. Emmerson and Robert I. Damper. 1993. Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application. *IEEE Transactions on Neural Networks* 4, 5 (1993), 788–793.
[158] Dhananjay S. Phatak and Israel Koren. 1995. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks* 6, 2 (1995), 446–456.
[159] David A. Medler and Michael R. W. Dawson. 1994. Training redundant artificial neural networks: Imposing biology on technology. *Psychological Research* 57, 1 (1994), 54–62.
[160] Troya Çağıl Köylü, Said Hamdioui, and Mottaqiallah Taouil. 2022. Smart redundancy schemes for ANNs against fault attacks. In *2022 IEEE European Test Symposium (ETS)*. IEEE, 1–2.
[161] Luis Alberto Aranda, Pedro Reviriego, and Juan Antonio Maestro. 2017. A comparison of dual modular redundancy and concurrent error detection in finite impulse response filters implemented in SRAM-based FPGAs through fault injection. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 3 (2017), 376–380.
[162] Sharon Hudson, R. S. Shyama Sundar, and Srinivas Koppu. 2018. Fault control using triple modular redundancy (TMR). In *Progress in Computing, Analytics and Networking*. Springer, 471–480.
[163] Yu Li, Yannan Liu, Min Li, Ye Tian, Bo Luo, and Qiang Xu. 2019. D2NN: A fine-grained dual modular redundancy framework for deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 138–147.
[164] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 27–32.
[165] Annachiara Ruospo, Gabriele Gavarini, Ilaria Bragaglia, Marcello Traiola, Alberto Bosio, and Ernesto Sanchez. 2022. Selective hardening of critical neurons in deep neural networks. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 136–141. DOI: http://dx.doi.org/10.1109/DDECS54261.2022.9770168
[166] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS One* 10, 7 (2015), e0130140.
[167] Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. 2020. Automated design of error-resilient and hardware-efficient deep neural networks. *Neural Computing and Applications* 32, 24 (2020), 18327–18345.
[168] Jun Zhang and Kim-Fung Man. 1998. Time series prediction using RNN in multi-dimension embedding phase space. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, Vol. 2. IEEE, 1868–1873.
[169] Weiqiang Liu, Chongyan Gu, Máire O'Neill, Gang Qu, Paolo Montuschi, and Fabrizio Lombardi. 2020. Security in approximate computing and approximate computing for security: Challenges and opportunities. *Proceedings of the IEEE* 108, 12 (2020), 2214–2231.
[170] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1211–1220.
[171] Lixue Xia, Mengyun Liu, Xuefei Ning, Krishnendu Chakrabarty, and Yu Wang. 2017. Fault-tolerant training with online fault detection for RRAM-based neural computing systems. In *Proceedings of the 54th Annual Design Automation Conference*. 1–6.

Received 28 June 2022; revised 25 December 2022; accepted 13 March 2023