Journal of Hardware and Systems Security
https://doi.org/10.1007/s41635-024-00144-8

Machine Learning-Based Security Evaluation and Overhead Analysis of Logic Locking

Yeganeh Aghamohammadi[1] · **Amin Rezaei**[2]

Received: 29 April 2023 / Accepted: 12 January 2024

Abstract
Piracy and overproduction of hardware intellectual properties are growing concerns for the semiconductor industry under the fabless paradigm. Although chip designers have attempted to secure their designs against these threats by means of logic locking and obfuscation, due to the increasing number of powerful oracle-guided attacks, they are facing an ever-increasing challenge in evaluating the security of their designs and their associated overhead. Especially while many so-called "provable" logic locking techniques are subjected to a novel attack surface, overcoming these attacks may impose a huge overhead on the circuit. Thus, in this paper, after investigating the shortcomings of state-of-the-art graph neural network models in logic locking and refuting the use of hamming distance as a proper key accuracy metric, we employ two machine learning models, a decision tree to predict the security degree of the locked benchmarks and a convolutional neural network to assign a low-overhead and secure locking scheme to a given circuit. We first build multi-label datasets by running different attacks on locked benchmarks with existing logic locking methods to evaluate the security and compute the imposed area overhead. Then, we design and train a decision tree model to learn the features of the created dataset and predict the security degree of each given locked circuit. Furthermore, we utilize a convolutional neural network model to extract more features, obtain higher accuracy, and consider overhead. Then, we put our trained models to the test against different unseen benchmarks. The experimental results reveal that the convolutional neural network model has a higher capability for extracting features from unseen, large datasets which comes in handy in assigning secure and low-overhead logic locking to a given netlist.

Keywords Logic locking · Obfuscation · SAT attack · Machine learning · Labeling · Decision tree · Convolutional neural network · Graph neural network

1 Introduction

Due to fabless manufacturing, it is critical for the semiconductor industry to protect hardware Intellectual Properties (IPs) from malicious threats. On one hand, the hardware design costs are high, and the designers need to protect their Integrated Circuits (ICs) from piracy. IC designers, on the other hand, are in danger of not getting enough revenue for their products due to unauthorized overproduction by untrusted third-party foundries. To overcome these malicious threats, an IC can be locked by adding extra gates controlled by secret key inputs [1–4]. In this case, the circuit works properly only when the correct key is being inserted; otherwise, it malfunctions. One naive attempt to attack the logic-locked circuits is to exhaustively search and check all the possible key inputs with the help of an activated IC bought off the market. While this brute-force attack is exponentially time-consuming with respect to the key size, there are more efficient oracle-guided attacks that can find the correct key in a relatively short time with the help of a Boolean satisfiability (SAT) solver. These attacks can find either an exact key [5] or an approximate key with low error [6, 7] but they may still become stuck due to a specific lock design or time and memory limits [8].

After proposing several oracle-guided SAT-based attacks, the logic locking research field has been growing with sequences of attacks [9–15] and defenses [8, 16–35]. While this game continues, the job of the defender requires

✉ Amin Rezaei
amin.rezaei@csulb.edu

Yeganeh Aghamohammadi
yeganeh@ucsb.edu

[1] University of California, Santa Barbara, CA, USA

[2] California State University, Long Beach, CA, USA

Published online: 22 January 2024

more effort since if a logic locking method is secure against all but one attack, the defender is doomed! Thus, one basic question becomes more critical than before: *how secure is a newly proposed logic-locked design?* Please note that from a hardware designer's perspective, we consider oracle-guided attacks, which are stronger than oracle-less attacks. If a scheme is secure against a strong attacker model, most likely it is also secure against a weaker model. Additionally, the ultimate goal of logic locking and obfuscation is to provide security for an efficient and precious design from piracy and overproduction, and it is implausible and cost-ineffective if the locking overhead is beyond a small, reasonable amount. So, considering the fact that there might be more than one locking method to secure the design effectively, another noteworthy question comes up: *how much overhead is imposed by locking?* In other words, a reliable framework is required to demonstrate which type of locking works best for each circuit with the minimum overhead.

In addition, recent machine learning (ML) attacks on logic locking demonstrate the presence of structural and functional leakage in state-of-the-art locking schemes, which has been disregarded by the traditional understanding of security [36]. While all of these initiatives are on the attacker side [37–45], no comprehensive study has been done on the opportunities that ML-based analysis might give for the hardware designers to protect their ICs against piracy and overproduction. We believe that with the help of ML, hardware protection becomes easier, more proactive, and more affordable. However, it can only do so if the underlying data gives a comprehensive view of the environment [46].

Thus, in this paper, we first build a multi-label dataset in both text and image formats by running different attacks on benchmarks locked with existing logic locking methods to evaluate the Error Rate (ER) of the reported key and compute the imposed overhead. Then, we propose an ML-based framework to assess the security of a given logic-locked circuit against various attacks. From a hardware designer's perspective, we aim to assign a secure yet affordable locking method to protect ICs in a fabless paradigm. Specifically, we examine the effectiveness of two ML models: the decision tree (DT) model, which extracts features from text data, and the convolutional neural network (CNN) model, which extensively finds features of various circuits from image-based data. With the help of resynthesis and data augmentation, the ML models can be trained on thousands of samples, gain high accuracy, and become noise-resilient.

In addition, recent studies [39, 44] have shown a promising direction in the advancement of logic locking "attacks" by employing graph neural networks (GNN). GNN is a strict generalization of CNN, yet it consumes more power and resources to run the algorithm, get trained, and predict the labels. In this study, we explore the opportunities that DT and CNN models would provide for improving logic locking "defenses." We also discuss the shortcomings of state-of-the-art GNN models in logic locking and refute the use of hamming distance (HD) as a proper parameter for key accuracy. The main contributions of this work are as follows:

- Discussing the shortcomings of current GNN models as well as the HD parameter in logic locking modeling and evaluation;
- Developing a multi-label security and overhead degree dataset consisting of more than 10,000 benchmarks locked with distinctive logic locking methods and tested under different oracle-guided attacks;
- Building and training a basic DT-based model for security evaluation of logic-locked circuits and an accurate CNN-based model for assigning a secure and low-overhead logic locking method to given circuits with suitable fairness checking and hyperparameter tuning;
- Testing the created ML models on unseen logic-locked benchmarks and evaluating the models' security and overhead prediction accuracy.

The rest of the paper is organized as follows: Section 2 explores the background and provides a literature review on logic locking attacks and defenses. Section 3 discusses the shortcomings of state-of-the-art GNN-based models in logic locking and argues the reason behind choosing ER as a more meaningful labeling parameter over HD. Then, Section 4 proposes two ML-based models for logic locking security evaluation and overhead analysis. Our extensive experimental result is given in Section 5. Finally, conclusions and future directions are discussed in Section 6.

2 Background and Related Works

In this section, we review the background of logic locking defenses categorized into pre-SAT and post-SAT methods and discuss the logic locking attacks grouped into oracle-guided SAT-based and ML-based attacks.

2.1 Logic Locking Defenses

Logic locking defenses can be divided into pre-SAT approaches such as XOR-based locking [1], MUX-based locking [2], AND/OR locking [3], LUT-based locking [4] and post-SAT approaches such as SAR-lock [16], Anti-SAT [17], TTLock [18], CAC [19], RND-cycle [20], R &D-cycle [21], BLE [8], ASSURE [47], and UNSAIL [48] methods. In all of these logic locking schemes, the correct key must be put in a tamper-proof memory or integrated in the circuit [49–51] soon after the fabricated ICs return from the foundry.

2.1.1 Pre-SAT Defenses

In XOR-based locking [1], the key bits can be matched with a combination of random buffers and inverters. After that, key bit controlled XOR gates are used to replace the chosen buffers and inverters. If an XOR gate is hiding a buffer, the correct key bit is "0" while if it is hiding an inverter, the correct key bit is "1."

In addition, in MUX-based locking [2], random signals are chosen and replaced with 2-1 MUXs with the real signal and random dummy ones as inputs and key bits as the selectors. The correct key here must select the terminal to which the real signal is connected and avoid the dummy signal.

The AND/OR locking strategy suggested in [3] inserts key-controlled AND/OR gates in place of signals with a predetermined imbalanced probability and a minimum slack.

The LUT-based locking [4] is being implemented to IC prefabrication. The goal is to separate the inputs from the outputs so that every path from inputs to outputs passes through a barrier. The key inputs are basically the values stored in the lookup tables.

2.1.2 Post-SAT Defenses

SAR-Lock [16] and Anti-SAT [17] are post-SAT locking solutions that secure the correct key from the attacker by increasing the number of Distinguishing Input Patterns (DIPs) that may be used to prune a wrong key. In this situation, the original SAT attack [5] will take exponentially more iterations to find the correct key with respect to the input size. In [18] TTLock is proposed to modify the design for exactly one input pattern. The modified design is then corrected using a key-controlled restore unit. Removing the restore unit results in a modified design instead of the original design in SARLock and Anti-SAT. Another approach called Corrupt-and-Correct (CAC) [19] corrupts the original design using hard-coded AND-trees and corrects the modified functionality using a generic correction unit.

In [20], random cycle insertion (RND-cycle) is presented, which inserts fake cycles in the circuit with two criteria. First, each cycle must have many entrance points. Second, each cycle must include at least two removable edges. In addition, dummy and real cycle insertion (R &D-cycle) [21] turns an acyclic combinational circuit to a cyclic circuit before cyclically locking it. First, for each real cycle, a 2-1 MUX is established, with one input connected to the real feedback signal and the other input being a random signal along the cycle's feed-backward route. In other words, picking one of the MUX's inputs causes the circuit to be cyclic, while selecting the other maintains it acyclic. The correct key bit must pick the real feedback in this scenario. Then some random signals are chosen, each of which serves as an input to multiple gates. After that, for each of those signals, an extra 2-1 MUX is inserted. Following that, for one of the MUX's inputs, a random dummy feedback is added via the feed-forward line, while the other input is linked to the original selected signal. The correct key bit must prevent the fake feedback in this case.

As an advanced post-SAT method, Bilateral Logic Encryption (BLE) [8], uses obfuscation and integrated locking on a sensitive component of a circuit. Using this method, the security impact, including the structural complexity and the logic complexity, gets transmitted to the entire circuit, whereas the performance overhead is less than the locking of the whole circuit.

As one of the most recent state-of-the-art locking methods, ASSURE [47] aims IP protection approach at the Register Transfer Level (RTL). This Electronic Design Automation (EDA)-tool-independent locking approach obfuscates the ICs before logic synthesis. By employing three different techniques, ASSURE obfuscates control branches, arithmetic operations, and constants, which leads to providing indistinguishable RTL designs.

Another recent strategy is called UNSAIL [48] which focuses on providing a combinational locking method with a low area overhead on a structure level to confuse ML attacks. By inserting unsuitable data during the learning stage of an ML attack, UNSAIL misleads the operation, and the ML model predicts the labels incorrectly. As one of the scopes of our work is to include the area overhead in the labeling process, we use UNSAIL in the verification stage. It is worth mentioning that both ASSURE and UNSAIL have considered an oracle-less attacking environment. Also, a holistic security diagnostic tool called Valkyrie is proposed [52] to identify vulnerabilities in the considered logic locking techniques from the standpoint of structural attacks. Our focus on this paper is, however, on oracle-guided logic attacks.

2.2 Logic Locking Attacks

Pre-SAT Hill Climbing attack [9] selects a random key candidate and progressively moves it closer to the correct key depending on observed test patterns guided by the hill climbing search. Randomly chosen key bits are toggled one by one at each iteration, and the current key combination is then subjected to a random test. To minimize disparities between actual and expected responses, a key bit value of "0" or "1" is selected. There are also oracle-guided attacks such as SAT [5], Double DIP [7], AppSAT [6], CycSAT [11], BeSAT [12], and IcySAT [14] as well as ML-based attacks such as GALU [40], SAIL [37], CutSAIL [42], UNTANGLE [43], OMLA [44], and GNNUnlock [39].

2.2.1 Oracle-Guided SAT-Based Attacks

The original SAT-based attack [5] repeatedly finds unique input patterns called DIPs, to exclude equivalence classes of keys. During this process, the attack finds an exact key value that matches all feasible input/output patterns. Although the SAT-based attack ensures finding the correct key in pre-SAT logic locking methods, finding the correct key may take exponential time in the case of post-SAT locking approaches.

To attack post-SAT locking approaches like SAR-Lock [16], Double DIP [7] is proposed to repeatedly find two DIPs (instead of one) in each iteration and prune incorrect keys.

Finding the key of a locked circuit considering the exactness limitation of the SAT-based attack could lead to a huge amount of memory and time consumption, which is not ideal for real-world problems. AppSAT [6] uses an approximate flow to find the probably approximate-correct key. Random sampling and a user-defined error threshold can be used to establish the degree of approximation.

In order to attack RND-cycle logic locking [20], an oracle-guided attack called CycSAT [11] is proposed. CycSAT assumes that there exists at least one correct key for which no structural cycle occurs in the circuit. The attack first calculates a formula, presuming the circuit has no sensitizable cycles, and then performs the original SAT-based attack on the constrained locked circuit.

While CycSAT uses structural analysis to find all the possible cycles in a cyclically locked circuit, BeSAT [12] pursues a behavioral method to unlock cyclic logic locking with the goal of reducing the missing cycle problem of CycSAT.

Finally, IcySAT [14], as another SAT-based attack on cyclic circuits, has been introduced that follows a cycle unrolling approach, contrary to CycSAT and BeSAT, which follow a cycle breaking strategy.

2.2.2 ML-Based Attacks

Recently, the application of Artificial Intelligence (AI) and especially ML in hardware security has attracted attentions [53–55], and some studies [37, 39, 40, 42–44] have proposed ML-based attacks on logic locking methods. While all of these ML frameworks are in favor of the attackers, no research has been conducted on the potential that ML-based analysis may provide for hardware designers to secure their ICs against piracy and overproduction.

In [40], an ML attack is introduced to unlock logic-locked circuits while reducing the run-time overhead of unlocking logic. With the help of genetic algorithms, circuit unlocking is converted into an optimization problem to find the key. At each stage, less probable key sequences are eliminated to gradually arrive at the correct key.

In [37], a structural ML attack calls structural analysis using machine learning (SAIL) retrieves the design of an obfuscated circuit on a gate-level structure. Using circuits as inputs, the work utilizes a random forest ML model for extracting features. This oracle-guided ML attack mainly aims at those structural obfuscating techniques that transform the circuits by small, local changes.

While SAIL mainly focuses mainly on the XOR-based locked circuits, CutSAIL [42] infers missing k-cuts from the neighboring logic. The work basically predicts the functionality of the embedded, missing parts of an obfuscated circuit. By using a proper neighbor encoding notation such as adjacency lists or adjacency matrices, the circuit information gets fed to a GNN model, thereby capturing the topology of the circuit.

In [43], researchers have presented an oracle-less link prediction attack based on GNN called UNTANGLE that learns the structure of gates in an obfuscated netlist. First, by mapping the key-extraction process as a link prediction problem, the attack model infers concealed links in the obfuscating blocks. Then, thanks to GNN, it learns the structure of the circuit, the features of the gates, and gradually the features of the links.

In addition, in [44], a GNN-based attack called Oracle-less Machine Leaning Attack (OMLA) is proposed, which employs subgraph classification to find the key bit values. OMLA explores the obfuscated netlist to extract small subgraphs for the key gates. The process of finding the subgraphs continues until the enclosing subgraphs capture the features related to the key bit values of the key gates. When it comes to labeling, OMLA considers the key bit value as the label for a subgraph. Then, each of the labeled subgraphs gets fed to a GNN model, letting it learn the characteristics associated with the circuit and predict the key bit values. This work is synthesis-tool independent, which makes it applicable to handling circuits synthesized by various scripts.

Unlike OMLA, GNNUnlock [39], utilizes GNN for node classification of circuits. The dataset for GNNUnlock is multiple obfuscated circuits of a benchmark with different key sizes. Using a set of edges and nodes to respectively represent wires and gates, the model uses adjacency matrices corresponding to the circuit as input to the GNN model.

3 Shortcomings of Current ML Models

In this section, we offer a critical view of the current GNN-based models in logic locking and show why HD is not enough to be considered a key accuracy metric. The work in [56] has also investigated other limitations of GNN models used for logic locking attacks and demonstrated how, by harnessing these limitations, the circuits can be protected against such attacks.

3.1 A Critique of GNN-Based Attacks in Logic Locking

Considering n, m, and k be the size of the input, output, and key respectively, we define the original circuit as $\mathbb{F} : \{0,1\}^n \rightarrow \{0,1\}^m$, and the locked circuit as $\mathbb{G} : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^m$ in which there is a k-bit correct key $K^* = (k_0^*, k_1^*, ..., k_{k-1}^*) : \{0,1\}^k$ such that $\mathbb{F}(X) = \mathbb{G}(X, K^*)$.

Although, graph representation preserves the topology of the circuit, using an undirected graph for netlist representation is one of the shortcomings of current GNN models because the inputs/outputs neighborhood of the netlist will be indistinguishable. In addition, current GNN-based attacks cannot effectively distinguish the difference between XOR-based controlled key gates and XNOR-based controlled key gates due to a lack of consideration of circuit functionality.

Let $K^a = (k_0^a, k_1^a, ..., k_{k-1}^a)$ be a k-bit reported key by the attack, the hamming distance (HD) of K^a and K^* can be defined as the sum of the bitwise XOR of the two keys:

$$HD(K^a, K^*) = \sum_{i=0}^{k-1} k_i^a \oplus k_i^* \; : \; \{0, 1, ..., k\} \tag{1}$$

Proposition 1 *GNN-based attacks can report an approximate key K^a of the locked circuit $\mathbb{G}$ in which $HD(K^a, K^*)$ is very small.*

Counterexample 1 We consider OMLA [44] as one of the GNN-based attacks, in which its prediction accuracy has been shown to be on average 80%. In means, for a reported key K^a, it is expected to predict almost 80% of the key bits correctly (i.e., $HD(K^a, K^*) = 0.2k$). If we replace all the XOR gates with XNOR in the benchmarks with XOR-based locking [1] and push the inverters to the fanouts, the new correct key will be the complement of the previous one. Figure 1 depicts an example of such a transformation on one key bit. However, the attack accuracy drops significantly to an average of 56% (i.e., $HD(K^a, K^*) = 0.44k$) which is not much better than reporting a random key.

The above counterexample implies that Proposition 1 is wrong and that GNN models are highly dependent on the specific gates in the circuit but not on their dependencies with each other. In other words, they cannot distinguish between two circuits with different topologies that have the same functionality.

Takeaway 1: GNN-based attacks on locked circuits with the same functionality but different structure do not necessarily outperform reporting a random key.

It is worth mentioning that to address some of the shortcomings of GNNs, in [57] obtaining a general and effective neural representation of circuits is discussed. In addition, in [58], a netlist representation learning framework is proposed to effectively acquire generic functional knowledge from netlists. Further, an automated framework is proposed in [59] that generates the data-flow graph for circuits such that the proximity in the embeddings indicates similarity between circuits. Also, in [60], a self-supervised netlist learning method is proposed that generalizes well using one-shot RTL of a design to recover the functionality of obfuscated designs.

3.2 A Critique of Using Hamming Distance as Key Accuracy

For a given key K of the locked circuit $\mathbb{G}$, we can define its error rate $ER(K)$ as the number of input patterns in which $\mathbb{F}(X) \neq \mathbb{G}(X, K)$, divided by all the input patterns (i.e., 2^n). Based on this definition $ER(K^*) = 0$. The important note here is that the ER value is intrinsically based on the functionality of the circuit, rather than its topology.

Proposition 2 *The smaller the $HD(K^a, K^*)$, the more similar the locked circuit $\mathbb{G}$ under K^a functions to the original circuit $\mathbb{F}$.*

Counterexample 2: Consider the locked circuit in Fig. 2 with a key size of $k-1$. We increase the key size to k by XORing one of the outputs with additional key bit k_{k-1}. In this case, there is a key K^a where just the key bit k_{k-1} is incorrect and all the other key bit values are the same as K^*. In other words, while the HD of K^a is very low (i.e., $HD(K^a, K^*) = 1$), the locked circuit $\mathbb{G}$ under K^a outputs differently than the original circuit $\mathbb{F}$ in 100% of the input patterns (i.e., $ER(K^a) = 1$).

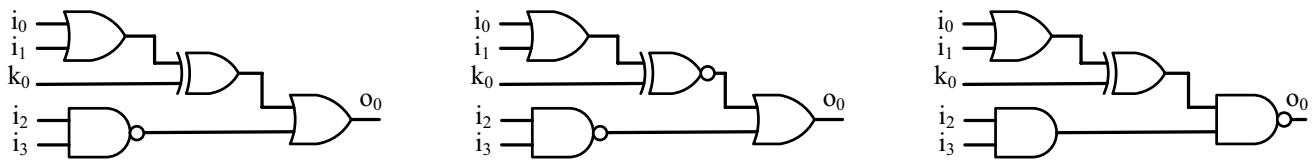


Fig. 1 Counterexample for the accuracy of GNN-based attacks

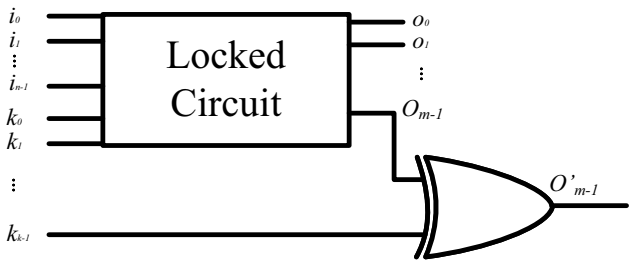


Fig. 2 Counterexample for hamming distance as a key accuracy metric

Therefore, Proposition 2 is wrong, and HD does not capture the functionality of the circuits and thus cannot be a good metric for key accuracy. We believe incorporating ER into the training dataset can be a better metric in this regard.

Takeaway 2: A key of the locked circuit with a small hamming distance to the correct key does not necessarily outperform a random key with a large hamming distance.

4 Logic Locking Defensive Frameworks

In this section, we utilize two distinct ML algorithms to extract features for evaluating the security degree of the locked benchmarks as well as assigning a secure and low-overhead logic locking method to a given circuit. The DT-based model examines the text dataset, while the CNN-based model extracts features from the image dataset.

4.1 MADELINE: DT Model for Locking Security Evaluation

In this section, we propose *MADELINE*, a multi-label decision tree based machine learning model for logic locking security evaluation. The process of using a DT-based model is categorized into three main phases as shown in Fig. 3. In the first phase, we gather and clean data, i.e., we run different attacks on distinct locking techniques, and based on finding the exact key, an approximate key, or no key, we label the data. Now that each locked benchmark has a specific label, we move on to the next phase, which is building *MADELINE*. In this phase, we preprocess the data and randomly allocate some to the training set and the rest to the testing and evaluation sets. With the help of hyperparameter tuning, we improve the accuracy of the model and prevent it from getting overfit. Last but not least, *MADELINE* is fully trained and ready to predict security labels of unseen locked benchmarks.

4.1.1 Data Gathering and Labeling for the DT Model

Some attacks are approximate in nature, meaning they find an approximate key rather than the exact correct key. To show whether or not the found key is exact, we refer to the ER of the key, which has been defined and discussed in Section 3.2. The attack is considered successful if the reported key has a low ER with respect to the input size. Also, some attacks perform well on some benchmarks while they cannot find the correct key for other benchmarks considering the time and memory limits.

We label data via two steps, as shown in Algorithm 1. First, we run various attacks on the benchmarks locked with different logic locking methods, approximately evaluating the correctness of the reported key (if any) using random test patterns and assigning a unique label for every benchmark under each attack. If the ER of the reported key is "1," then the locked benchmark is considered "safe." Also, if a timeout happens, i.e., if it takes more than one day with no reported key, the benchmark is labeled "safe" too. If the attack returns a key with an error rate of "0.3" to less than "1," the locked benchmark is considered "semi-safe," implying that the locking method can protect the circuit to some extent. If the found key is equivalent to the correct key or the ER of the reported key is between "0" and "0.3," it is labeled as "unsafe." It should be noted that some attacks have higher priorities compared to others. For instance, if the SAT attack [5] could find the key, the locked benchmark is definitely "unsafe," but if AppSAT [6] finds a key, we should test the key and find the approximate ER of the key.

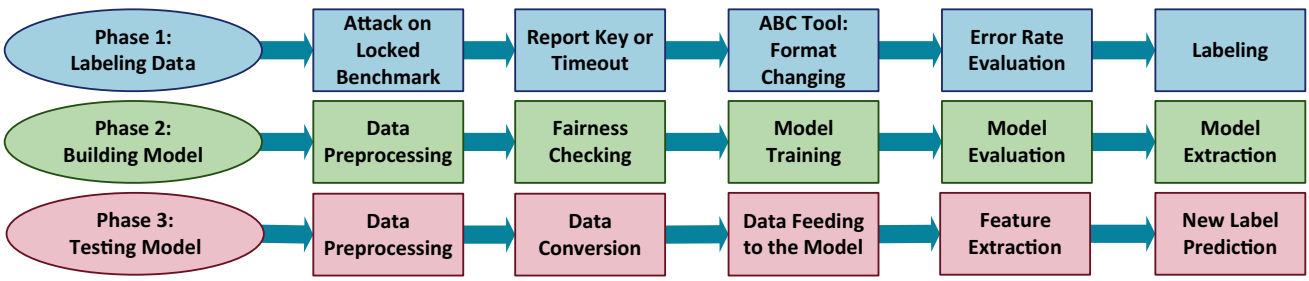


Fig. 3 MADELINE framework

Then, in order to find the final label of each logic-locked benchmark, if at least one of the attack methods could find the key, the logic-locked benchmark is "unsafe." If none of the labels are "unsafe" but at least one of the labels is "semi-safe," the benchmark is labeled as "semi-safe," and if all of the labels are "safe" the whole scheme is "safe." With the data being ready, we can move on to phase 2, which is building and training *MADELINE*.

Algorithm 1 Labeling algorithm

```
Input: Sets of locked benchmarks B and attacks A
Output: Set of labeled locked benchmarks L
for ∀ locked benchmarks ∈ B do
    for ∀ attacks ∈ A do
        if TimeOut() ‖ Error(Key) = 1 then
            Add a "safe" label to the benchmark
        else if 0.3 ⩽ Error(Key) ¡ 1 then
            Add a "semi safe" label to the benchmark
        else
            Add an 'unsafe" label to the benchmark
if ∃ an "unsafe" label then
    Add a final "unsafe" label to the benchmark
else if ∃ a "semi safe" label then
    Add a final "semi safe" label to the benchmark
else
    Add a final "safe" label to the benchmark
```

Algorithm 2 Decision tree algorithm

```
Input: Labeled data
Output: Decision tree
for ∀ Labeled Benchmark ∈ Labeled Data do
    gain ← 0
    split ← null
    entropy ← Entropy(Attributes)
    for ∀ Attributes att ∈ Labeled Data do
        infoGain ← InformationGain(att, entropy)
        if infoGain > gain then
            gain ← infoGain
            split ← att
```

4.1.2 Building the DT Model

MADELINE identifies the strategy of all the attributes of labeling, using an ML model based on DT [61] as shown in Algorithm 2. To build the DT, we define three major parts: a decision node gets the unlabeled data; a chance node predicts the probability of the potential label; and an end node represents the final labeling decision for the input data.

To build the DT, we should use two functions: *Entropy* and *InformationGain* [62]. In building a DT, our goal is to find the best-split option for a given blended data. This splitting process continues until the data is not mixed anymore. The *Entropy* function is being used in evaluating disorder and learning how blended a series of data is. At each stage, if the splitting point lowers the chaos of the dataset, entropy decreases.

During the splitting process, we have two sides, and for each side, we need to measure the entropy. Then, by subtracting the previous entropy from the current entropy, the *InformationGain* function will learn how well the splitting is, i.e., how much information each side gained. A positive result means the split learned something, which leads to lower entropy.

As nodes are being defined, the tree is ready to be trained so that it can learn features of the labeled data. Well-training a machine learning model requires some provisions. First, data allocation should be fair, i.e., data distribution in the train set and test set should cover almost all the features. Second, data proportion to the train and test sets should be in a way to avoid overfitting and underfitting. Overfitting refers to the term when a model works well on the training dataset but poorly on the test dataset. Underfitting happens when a model works poorly on both the train and test datasets.

To avoid the aforementioned problems, first, we perform fairness checking. Fairness checking is an important step in allocating data to the training, test, and evaluation sets. It assures that the data distribution among all the benchmarks and locking methods is as unbiased as possible and at least one benchmark with a certain locking exists in all the datasets. Although data allocation to each set is random, we preprocess the data at each set to guarantee fairness. If we skip the fairness checking step, it is possible that the model will not learn some data features. In this situation, the training accuracy would be reasonable, as the model is working well on the learned features, but the test accuracy could get noticeably lower as some features in the test set may not have been examined in the training phase.

Then, we perform hyperparameter tuning. During the hyperparameter tuning, we select the best set of hyperparameters to make sure the model is working properly in both the training set and the test set. To achieve this goal, we compare the accuracy of the training and testing sets using various hyperparameters to find the best hyperparameter set, optimize the model, and get the highest possible accuracy of the model, both in the training phase and the test phase. The hyperparameters of a DT include maximum depth, minimum split samples, minimum leaf samples, and maximum features. Maximum depth is an integer that defines the maximum intended depth of the DT. The greater the maximum depth, the more complicated the tree. With a low depth number, models do not have enough freedom to learn the features of the data. For instance, if we have ten labels for a dataset, the maximum depth of one would not help the tree learn enough features about the data. Minimum split samples specify the minimum number of samples required to split an internal node, and minimum leaf samples specify the minimum number of samples required to be at a leaf node. Choosing a low number for the two latter parameters helps the model differentiate well between samples. Maximum features refer to the number of features to consider when looking for the best split. Among the mentioned hyperparameters, we set the minimum number of leaf samples to be one, as we want the end nodes to have

one sample, and we do not set any particular number for the maximum features so that the model will consider all features available to make the best split.

To get the best accuracy from the model, we assign different values to the minimum sample size and maximum depth, to learn which combination helps the model get the highest possible accuracy, both with the training set and the test set. As we have three labels as "safe," "semi-safe," and "unsafe," we get the accuracy of the model with the maximum depth being set as one and two. Moreover, we allocate a different percentage of the data to the train set and test set, to get the desired model efficiency. Finally, as we need to evaluate the effect of minimum split samples, we examine the model with split samples equal to the number of labels and equal to the test size.

4.1.3 Testing the DT Model

During the training process, the model extracted features from the benchmark, and related the assigned label to that. Once the model is trained, learned all the features, and tested well on the train set and test set, it is ready to predict labels of unseen data.

To test the model, we need to follow the following steps. The first step of the testing phase is data preprocessing. As the data is new and has not been included in the training phase, we need to fully examine it to make sure the structure is the same as the training dataset and known to the model. For example, if the new benchmark contains new gates or components that did not exist in the training set, the model cannot predict the correct label. In such cases, if the new data contains unknown components, we should re-train the model with benchmarks containing those specific components.

Secondly, we should convert the new benchmarks to the same format as the training dataset. Next, as soon as the data is ready, we can feed it to our model, let it extract features, and compare those with the learned features of the training dataset. Based on the comparison of features, the model can predict the label of the new benchmark. This process repeats for each attack method, i.e., the model examines the new data under all the predefined attacks and extracts labels for them. Finally, the model evaluates all the attack labels of the benchmark to give the final label of the locking technique.

4.2 *CoLA*: CNN Model for Low-Overhead Locking Assignment

In this section, we propose *CoLA*, a convolutional neural network logic locking assignment model that finds a low-area-overhead secure locking method for a given netlist. First, we gather data and use the area overhead and the ER of each locked benchmark to label the data and make them ready for training. Then, we train *CoLA* on the augmented labeled data in order to extract their features. Finally, the trained model is ready to assign a low-overhead logic locking method to new, unseen data. Moreover, extracted weights of *CoLA* can be used for other CNNs with different structures. The *CoLA* framework is shown in Fig. 4.

4.2.1 Data Gathering, Labeling, and Augmentation

When it comes to training the neural network, a suitable dataset is needed so that the CNN model can learn many features of the circuit for a high-accuracy prediction. Data labeling has two phases. First, we need to find the ER of each locking technique for each benchmark. Second, we need to export the area overhead of the locked benchmarks. Based on the fact that not all the low-overhead locking methods can provide a secure design, we need to define a parameter that relies both on the area overhead and the security of the locking method.

Key Correctness Value (KCV): Approximate attacks find keys with an ER between 0 and 1, which has been calculated by random sampling of the oracle and the locked benchmark under the reported key. Along the same lines, we define a reported key's *KCV* as $1 - ER$. From the defender's perspective, the higher the *ER* (i.e., the lower the *KCV*), the more secure the locking technique.

Locking Area Overhead (LAO): We define LAO as the area overhead of the locking technique on the original circuit.

To allocate a parameter that mutually considers the effects of KCV and LAO, we define Low-overhead & Secure Label (LSL) as the following:

$$LSL = \alpha LAO + (1 - \alpha)KCV \tag{2}$$

where α is the weighted coefficient with an amount between 0 and 1 to put emphasis on the key correctness side or the

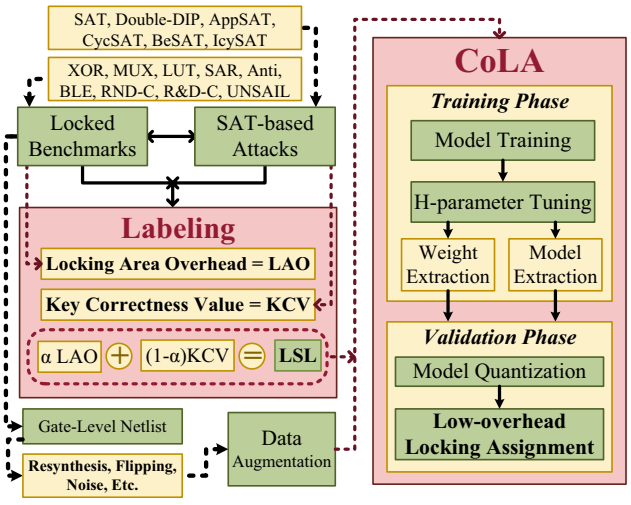


Fig. 4 *CoLA* framework

area side. Moreover, since acceptable costs must be defined based on customer and industry needs, LSL provides this flexibility. Based on Eq. 2, among all the predicted labels of a model, the lowest LSL is the best one, which means that the chosen locking technique has a trade-off between the low KCV (i.e., high security) and the low LAO (i.e., low area). As a circuit has multiple ERs because of the usage of different attack methods, we use the lowest ER for Eq. 2 to account for the highest vulnerability of a locking method against any of the attacks.

While thousands of samples are needed to efficiently train a CNN, publicly available logic-locked benchmarks are limited. As our CNN model needs image data to get trained, we can use different layouts of each benchmark to augment our data. To do so, we convert the .BENCH benchmarks into Verilog using the *ABC* tool [63]. Then, using different routing settings, we export 15 different layouts for each benchmark in terms of the structure of the layout, and the positions of the elements. As in our work, the structure of the layout matters, we use different layouts for the same functionality with the help of various resynthesis options. The different layouts are, including but are not limited to, showing simplified logic, grouping all related nodes, showing registers without fan-outs, and enabling global net routing. As shown in Fig. 5, to get even more data, we apply data augmentation techniques available in *Keras* [64] such as noise injection, random brightness, random flip, and rotating, to name a few [65]. As a result, we can get thousands of different layouts as input data to *CoLA*. This data augmentation helps our model to preserve its accuracy in case of noise injection, such as various gate positions.

The CNN model needs a training dataset and a testing dataset to get trained and tested. With the train dataset, the CNN model learns features, and with the test dataset, it tests the model's functionality. We randomly allocate 10% of the gathered data to the test set and the rest of it to the train set. The original input size of each image data ranges from 800×800 pixels in smaller benchmarks to 4000×4000 pixels in larger benchmarks. To use a decreased size of data suitable for the memory and processing resources available to us, we convert all the images to a size of 250×250 pixels, which is small enough to feed to the model, and large enough to preserve the structure and be readable for the model.

4.2.2 Training the CNN Model

A CNN is a framework that generally gets applied to explore visual data in the form of images. CNNs commonly use the shared-weight architecture of the convolution filters that slide along input data images with a pre-defined depth and provide a set of extracted features known as feature maps. A CNN's key benefit is that, if defined properly without overfitting and underfitting, it can adapt well to the dataset and give pretty accurate results on the unseen data as well.

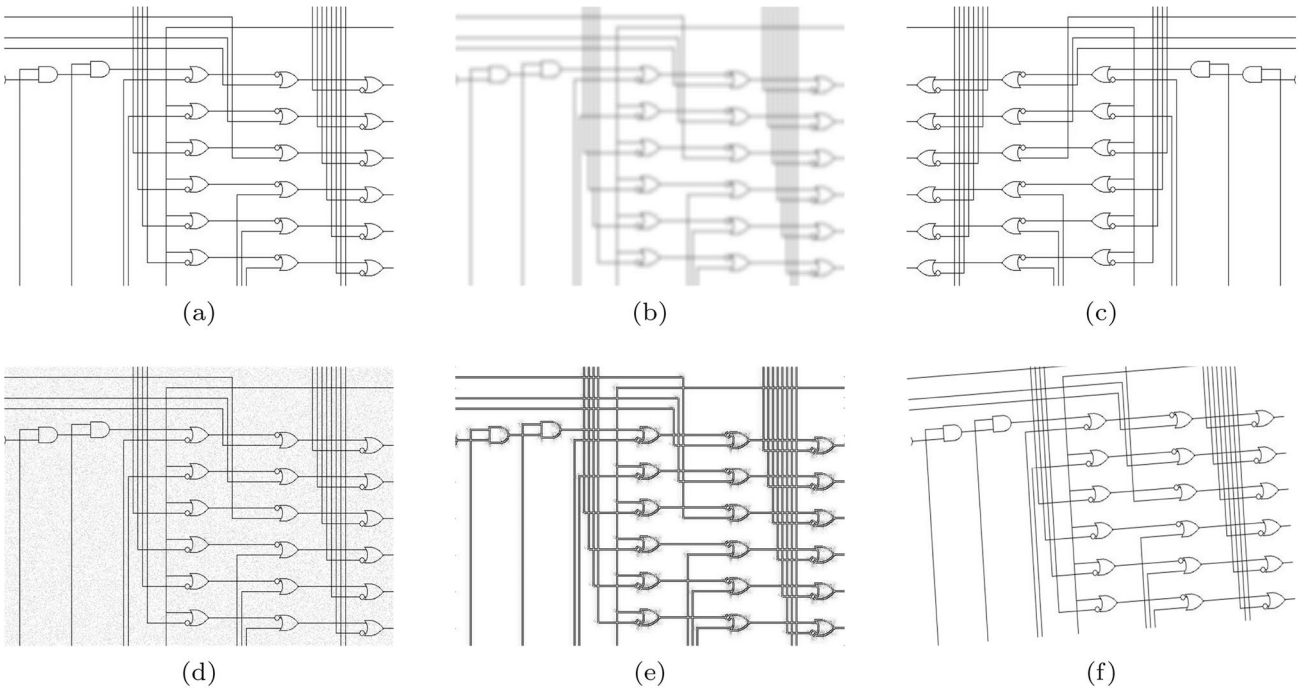


Fig. 5 Visual examples of data augmentation **a** original, **b** blur, **c** flip, **d** noise, **e** sharpen, and **f** rotation

Each CNN model should have enough nodes in each layer to understand features comprehensively. Also, the CNN model should be deep-enough, i.e., should have enough numbers of layers, to extract as many features as possible. In this work, we structure a CNN with five convolution layers for feature extraction, each following with a maximum pooling layer, and three dense layers, also known as fully connected layers, for classification. The number of nodes in the last dense layers is equal to the number of labels. Each convolution layer except the last one uses the LeakyReLU activation function with an alpha parameter equal to 0.01. LeakyReLU returns all the positive numbers to their own amount, and all the negative numbers to 0.01 of their amount. The accuracy of the model, in the training and validation phases, depends on many aspects such as the size of the dataset, the number of layers, and the size of the pooling layers. With the help of hyperparameter tuning, i.e., increasing the number of weights, changing the size of the pooling layers, and increasing the number of layers, we can increase the model's accuracy while avoiding overfitting. The structure of *CoLA* and its layers is shown in Fig. 6.

4.2.3 Evaluating the CNN Model

The training phase is an offline phase, which means users have access to capacious memory as well as enough timing. So, training a roughly large model, like *CoLA*, will not cause any timing or resource problems. But when it comes to the validation phase, which is an online phase, the size of the model and dataset affect the execution time, as well as the consumed memory. So, we propose to use a quantized version of *CoLA* for the validation phase. Hence, we can get high accuracy of the model, while consuming fewer resources. A quantized model uses less memory and processes the computation faster.

In this regard, if the majority of the numbers fall within the range we demand, we will use a lower bit representation to represent input, weights, and feature maps of the CNN model. For instance, if the original CNN model uses p bit numbers to compute the result, the quantized version uses q bit numbers where $q < p$. A CNN uses multiplication and addition to compute the results. So it is possible that the sum and product of two q bit numbers represent numbers larger than q bits, as the sum of two q bit numbers is a $q + 1$ bit number, and the product of two q bit numbers is a $2q$ bit number. To avoid this issue and keep the CNN model quantized to q bit, we use the quantization function at the product of each computation process and before feeding the number to the next stage of computation In the case of overflow and underflow in the quantization process, we assign the highest and lowest range demand, respectively.

5 Experimental Results

In this section, we study *MADELINE* and *CoLA*, discuss each methodology, examine the results, and discuss the advantages and disadvantages of each model. We implement *MADELINE* and *CoLA* on an Intel Core i7-10,750 H CPU, with a RAM size of 16 GB. To provide appropriate data for both of the models, we first need to find the ER of the reported keys for each benchmark using various attack algorithms. Then, the dataset process splits into two separate paths as the ML models work with different types of data. To create a labeled dataset, we use combinational circuit benchmarks of ISCAS'85 [66] and MCNC'91 [67], and apply different locking methods on each benchmark. Table 1 shows all the benchmarks and locking techniques we use as our data, as well as the gate and key size of each one. While small benchmarks such as "apex4" are generally

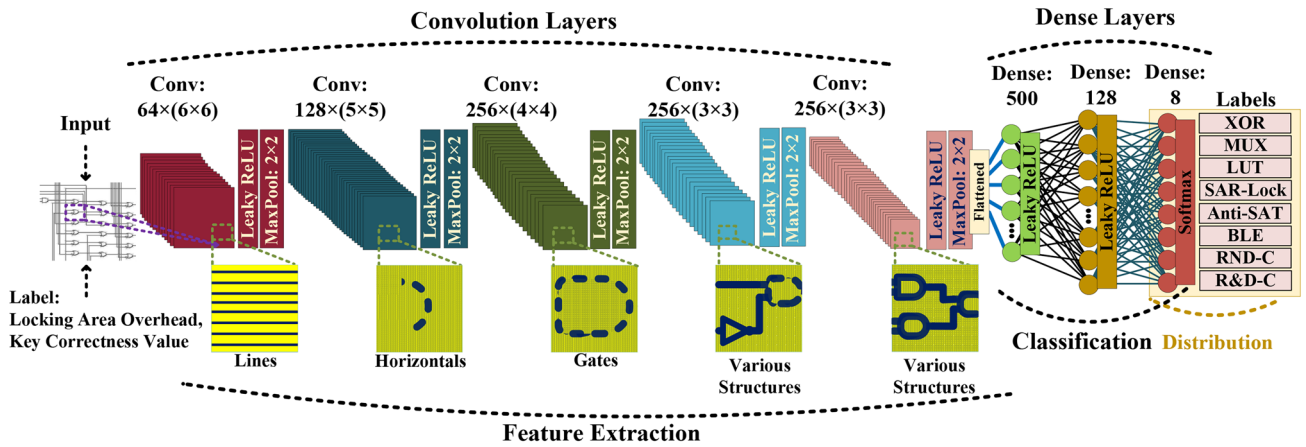


Fig. 6 *CoLA* structure

Table 1 Logic locking benchmarks (#In, #K, and #G are the number of primary inputs, key inputs, and gates, respectively)

Bench	#In	XOR-based		MUX-based		AND/OR-based		LUT-based		SAR-lock		Anti-SAT		BLE		RND-cycle		R &D-cycle	
		[1]		[2]		[3]		[4]		[16]		[17]		[8]		[20]		[21]	
		#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K
apex2	39	643	31	642	32	642	32	1780	292	644	31	687	38	713	40	630	20	670	40
apex4	10	5628	268	5628	269	5629	269	18,441	3356	5633	268	5381	10	5388	10	5380	20	5420	40
c432	36	170	8	170	10	140	10	1101	184	168	8	233	36	254	36	180	20	220	40
c499	41	212	10	214	12	214	12	1891	288	212	10	291	44	310	42	222	20	262	40
c880	60	404	19	385	22	405	22	1831	3112	403	19	504	60	536	60	403	20	443	40
c1908	33	925	44	734	46	926	46	1586	200	928	44	945	32	968	34	900	20	940	40
c2670	233	1253	60	1264	71	1264	71	3535	540	1257	60	1038	71	1781	234	1654	20	1253	40
c3540	50	1512	76	1502	76	1755	86	5026	836	1511	76	1634	83	1803	52	1689	20	1729	40
c5315	178	2427	115	2113	124	2431	124	7841	1176	2424	115	2307	134	1495	178	2327	20	2367	40
dalu	75	2418	115	2417	119	2417	119	4875	640	2436	115	2447	74	2491	76	2318	20	2358	40
des	256	6804	324	6809	336	6809	336	17,879	2856	6804	324	6550	38	7116	256	6493	20	6533	40
ex5	8	1109	53	1108	53	1108	53	4627	888	1109	53	1072	8	1078	8	1075	20	1115	40
i4	192	360	17	365	27	365	27	1538	272	355	17	527	94	821	192	358	20	398	40
i7	199	1384	66	1391	76	1391	76	4921	908	1389	66	1340	12	1818	200	1335	20	1375	40
i8	133	2589	123	2594	130	2594	130	8144	1348	2598	66	2533	34	2802	134	2484	20	2524	40
i9	88	1089	52	1091	56	1091	56	3477	608	1092	52	1088	26	1258	88	1055	20	1095	40
k2	46	1908	91	1907	93	1908	93	4482	620	1906	91	1908	46	1933	46	1835	20	1875	40
seq	41	3697	176	3697	178	3697	178	10,829	1848	3700	176	3600	40	3627	42	3539	20	3579	40

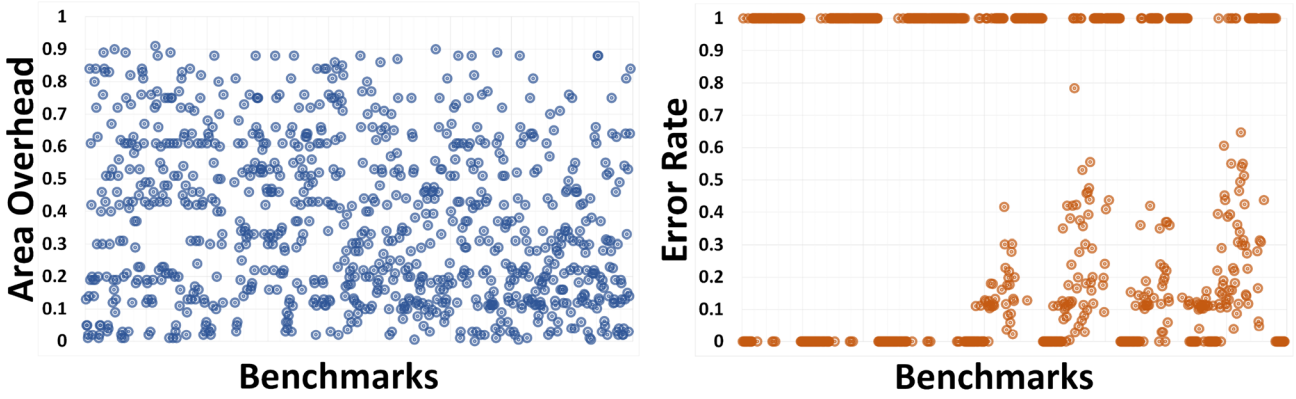
unsafe regardless of the chosen logic locking method, the ML model can benefit from them by learning that no logic locking method is by default secure, and even if a secure locking is used on a small circuit, the circuit is still unsafe because its functionality can be revealed by brute-force analysis of the activated IC.

5.1 *MADELINE* Evaluation

MADELINE's dataset is in the form of text (i.e., .BENCH files), so we create a dataset with the labels "safe," "unsafe," and "semi-safe," each of which refers to its specific ERs. The term "safe" refers to an ER equal to 1, "unsafe" refers to an ER between 0 and 0.3, and "semi-safe" refers to values between the ranges "safe" and "unsafe." We examined traditional pre-SAT locking methods such as XOR-based locking [1], MUX-based locking [2], AND/OR-based locking [3], and LUT-based locking [4] as well as post-SAT point function methods such as SAR-Lock [16] and BLE [8], and cyclic methods such as RND-cycle [20], and R &D-cycle [21].

If SAT [5], Double DIP [7] and CycSAT [11] attacks take more than one day to find the key, then we consider it a time-out, which means the key cannot be found, and thereby the locked benchmark is "safe" under these attacks. For AppSAT [6], BeSAT [12], and IcySAT [14], we used the *NEOS* suite [68] with the default setup. For Hill Climbing [9], the iteration limit is being set to 1200.

To extract the ER of a reported key, we test random samples equal to the logarithm of the input size. At this stage, we use the ABC tool [63] to convert the .BENCH files into .V files to be able to simulate and examine the oracle and locked benchmarks using ModelSim. To keep the results fair, we impose the same test set on the reported key of a locked benchmark for different attacks. Inputs get chosen randomly, but when we select a random input set on one locked benchmark, we keep applying the same input to all other logic-locked versions of that benchmark. For a given benchmark, the locked version and oracle will represent the same output if the inputs are the same and the applied key is correct. During sampling and comparing the waveforms, we record the approximate error of the key. When the ER is found, the locked benchmark is ready to be labeled. Figure 7a shows the error rate distribution of the dataset. We want to emphasize that an imbalanced ER in the dataset helps *MADELINE* learn the security of a logic-locked circuit better and extract distinctive features from the logic-locked circuits in which ensemble attacks can report an exact key (i.e., ER=0) or no key (i.e., ER=1). From a decision-making perspective, there is no "semi-safe" logic-locked circuit. However, we consider "semi-safe" labeling to give the user insights on a given locked circuit that is currently not safe, but it may be improved to become safe.



(a) (b)

Fig. 7 Dataset distributions **a** error rate **b** area overhead

Examples of the labeling are given in Tables 2 and 3 which show the ER and final labeling decision of two benchmarks with different locking methods. As shown in the table, if one attack could find the exact key, the ER equals 0, which means "unsafe" in our labeling technique. After the ER of all attacks is ready, we will define the final decision of the locking method on the benchmark based on the ER of the reported key by each attack. Based on the data in Tables 2 and 3, a benchmark could get different labels based on the locking technique and the attack method. For example, benchmark "c1908" under the LUT-based locking with 1586 gates and a key size of 200 bits is being labeled as "unsafe," whereas the same benchmark under the BLE benchmark with 968 gates and a key size of 34 bits is being labeled as "safe" because none of the attack methods could find the key. It is worth mentioning that some attacks could not find the key to a locked benchmark. For instance, SAT and Double DIP can not find the key of RND_C and R &D_C locking methods, hence their ERs are 1, known as "safe" for those attacks.

After creating the dataset, we build *MADELINE* in *PYTHON* and test it to find out the accuracy of the model. The accuracy of the training phase and testing phase can be affected by several hyperparameters. Figure 8 shows the accuracy results of *MADELINE*, based on various dataset proportions and different hyperparameters. Specifically, we investigate the accuracy of the model based on allocating 60% to 90% of the data to the train set, depth sizes of 1 and 2, and two different minimum sample split sizes. A general comparison of the accuracy gained with different minimum sample splits shows that a minimum sample split set to the size of the label works better than setting it equal to the size of the test set and it can help the model gain up to 13.54% total improvement in the training accuracy and 15.38% in the testing accuracy. A comparison between the results in Fig. 8 shows that the depth of the tree matters when the data allocated to the train set is less than 70%. That being said, if we allocate enough data to the train set, we do not have to worry about the complexity of the tree. Moreover, using appropriate hyperparameters, by allocating more data to the train set, we help the model learn features better, thereby achieving higher accuracy with the test dataset.

To evaluate the model and predict labels of the new, unseen data, we trained our model with 90% of the dataset,

Table 2 *MADELINE*: Sample labeling for apex4

Bench	SAT [5]	D-DIP [7]	CycSAT [11]	AppSAT [6]	Hill [9]	IcySAT [14]	BeSAT [12]	Label
SAR	0	0	0	0.002	0.002	0.054	0.002	Unsafe
AND/OR	0	0	0	0.033	0.421	0.42	0.38	Unsafe
LUT	0	0	0	0.31	0.531	0.35	0.36	Unsafe
XOR	0	0	0	0.1	0.289	0.36	0.273	Unsafe
MUX	0	0	0	0	0	0	0	Unsafe
BLE	0	0	0	0	0	0.044	0	Unsafe
RND_C	1	1	0	0	0	0.026	0	Unsafe
R &D_C	1	1	1	1	1	0.011	0	Unsafe

Table 3 *MADELINE*: Sample labeling for c1908

Bench	SAT [5]	D-DIP [7]	CycSAT [11]	AppSAT [6]	Hill [9]	IcySAT [14]	BeSAT [12]	Label
SAR	0	0	0	0	0.017	0	0.015	Unsafe
AND/OR	0	0	0	0	0.012	0	0.393	Unsafe
LUT	0	0	0	0.075	0.129	0.072	0.34	Unsafe
XOR	0	0	0	0	0	0	0	Unsafe
MUX	0	0	0	0	0.192	0	0.148	Unsafe
BLE	1	1	1	1	1	1	1	Safe
RND_C	1	1	0	0	0	1	0	Unsafe
R &D_C	1	1	1	1	1	0.41	1	Semi

as it helps the model get the highest possible accuracy without any overfitting. In Table 4, we provided some of the prediction results on unseen data. We used completely new data to make sure the model was unfamiliar with the structure of each benchmark. We also included combinational versions of two benchmarks from ITC'99 [69] to check the effectiveness of *MALEDINE* on a completely new dataset.

We performed attacks on these locked benchmarks and found the labels based on our labeling method. Then, we fed the unlabeled data to the model so that it could predict the label based on what it learned previously. The strong majority of the predicted labels are the same as actual labels, which means that the model learned most of the features of the training data and is working properly on unseen benchmarks. Overall, *MADELINE* has 99.01% of prediction accuracy.

The model can benefit from the hyperparameter tuning in the prediction phase, too. Not only does hyperparameter tuning improve the model's accuracy and prevent overfitting, it also helps reduce the model execution time for the prediction. In Table 5, we can see the effect of the mentioned parameters on the prediction execution time. For instance, if we use a complex tree, with a depth of 2 and the sample split size equals to the size of the test set, it takes 141 milliseconds for the model to predict the label, whereas for a

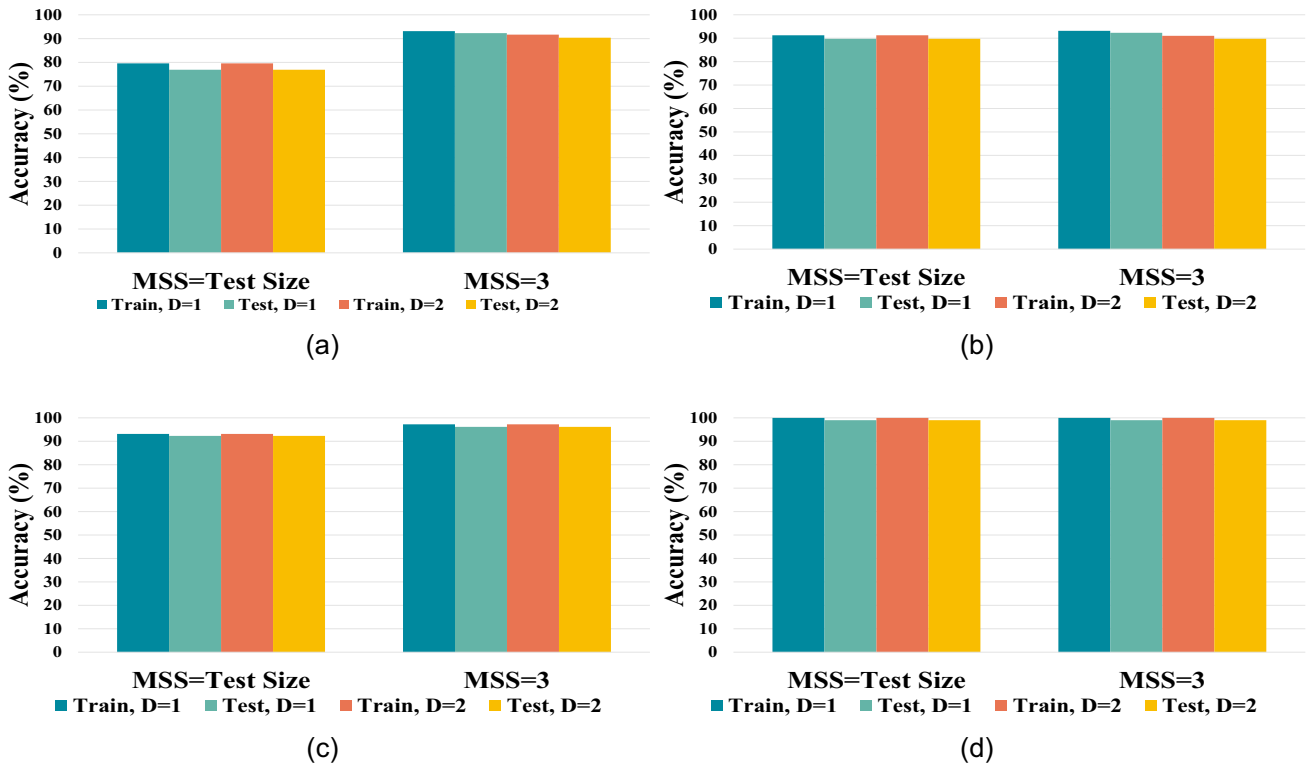


Fig. 8 *MADELINE* training & test set accuracy with different hyperparameters. D, depth; MSS, minimum sample split. **a** Train set = 60%, **b** train set = 70%, **c** train set = 80%, and **d** train set = 90%

Table 4 *MADELINE*: Unseen samples for label prediction

Bench	Lock	#In	#G	#K	Predict	Actual
c1355	SAR	41	693	137	Unsafe	Unsafe
c1355	SAR	41	837	273	Unsafe	Unsafe
ex1010	AND/OR	10	6335	1269	Unsafe	Unsafe
ex1010	AND/OR	10	7604	2538	Unsafe	Unsafe
ex1010	LUT	10	17,918	3256	Unsafe	Unsafe
c3540	LUT	50	5026	836	Unsafe	Unsafe
c7552	XOR	207	3877	351	Unsafe	Unsafe
c7552	XOR	207	5311	1756	Unsafe	Unsafe
c2670	MUX	233	1716	713	Unsafe	Unsafe
c7552	MUX	207	4880	1860	Unsafe	Unsafe
b14_C	MUX	227	9433	621	Unsafe	Unsafe
ex1010	BLE	10	5094	10	Unsafe	Unsafe
c6288	BLE	34	2504	34	Safe	Safe
b19_C	BLE	6666	213,520	6666	Safe	Safe
c3450	RND_C	50	1689	20	Semi	Semi
c7552	RND_C	207	3532	20	Semi	Semi
c3540	R &D_C	50	1729	40	Safe	Semi
c7552	R &D_C	207	3572	40	Safe	Safe

less complex tree, with a depth of one and a sample split size equals to the number of labels (i.e., 3), it takes 66 milliseconds for the model to predict the label. In this case, the model has the same accuracy but is 53.2% more efficient in execution time.

5.2 *CoLA* Evaluation

As *CoLA* is a CNN model, it gets trained on data in the form of images, compared to GNN models, which get data in the shape of graphs. Contrary to GNN, the CNN data is independent of the circuit size. A CNN model uses the same sizes of images to get trained, which may sound tricky at first because a fixed-size image must provide enough information about circuits with different sizes. In this regard, we should find a proper image size that is computation-friendly so that we can squeeze a large circuit into it while making sure not to lose circuit information. However, on the other hand, this could be helpful to the training resource usage because no matter what the size of the circuit is, the CNN data size is fixed.

Table 5 *MADELINE*: Prediction execution time for c6288 locked with BLE, based on the hyperparameter tuning

Min sample split	Max depth	Prediction time (ms)
Size of test set	2	141
Size of labels	2	88
Size of test set	1	91
Size of labels	1	66

In addition, a GNN model works properly at predicting the key only if it receives the features of the benchmarks before training. This makes the model limited to the structure of the circuit, and the model will not be able to distinguish resynthesized versions of a circuit. However, by feeding various structures of a benchmark to the CNN model, the model learns the features from scratch and categorizes resynthesized versions of a benchmark in the same group.

To extract data in the form of images, we used the web edition of Intel Quartus II. To create a labeled dataset, we examined different traditional pre-SAT locking methods such as XOR-based locking [1], MUX-based locking [2], and LUT-based locking [4] as well as post-SAT methods such as SAR-Lock [16], Anti-SAT [17], BLE [8], RND-cycle [20], and R &D-cycle [21]. We recorded the LAOs by comparing the area of each locked benchmark with its original version. To gather the ERs, we ran different attacks [5–7, 11, 12, 14] on the locked benchmarks and chose the minimum ER among the reported keys for each benchmark. The setup for SAT [5], Double DIP [7], and CycSAT [11] attacks is the same as the default setup, and we consider it a timeout with an ER of 1, if the key cannot be found in one day of running. For all the other attack methods (i.e., AppSAT [6], BeSAT [12], and IcySAT [14]), we used the *NEOS* suite [68] with the default setup. Then, we used Eq. 2 to assign "LSL" labels to each benchmark with $\alpha = 0.5$ which means that both area overhead and security degree are considered to be equally important. The distribution of the area overhead in our dataset is shown in Fig. 7b. Finally, we converted the benchmarks to image netlists and augmented the gathered data to more than 10, 000 samples using the approaches discussed in Section 4.

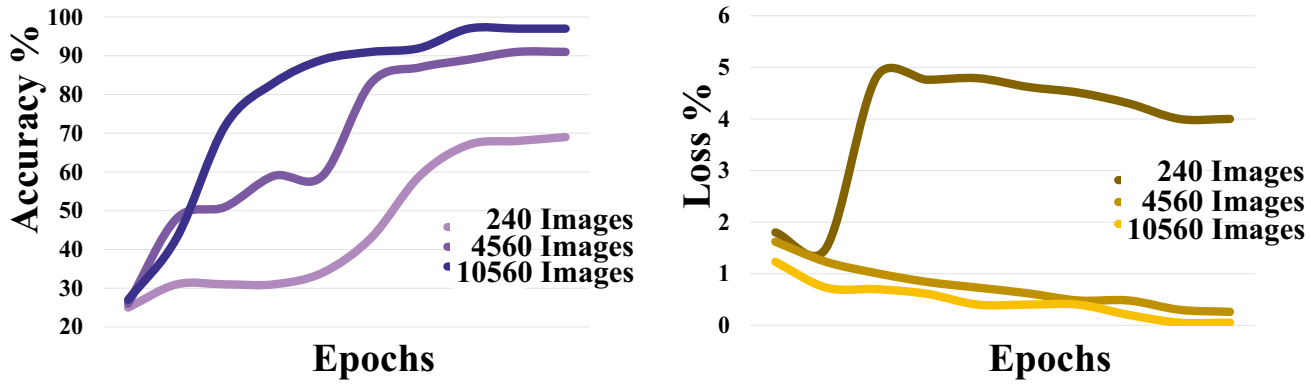


Fig. 9 *CoLA*: Validation accuracy and loss per epoch

After creating the dataset, we built *CoLA* using *Python* and the *Tensorflow* package, trained it, and extracted the features of the image-shaped benchmarks. Then, a quantized version of *CoLA* is used to get validation computations done fast. The accuracy of the training phase and testing phase can be affected by several hyperparameters. Using *Keras* [64] hyperparameter tuning, we increased the accuracy of the training and validation phases with a loss value of below 0.1% and without overfitting or underfitting. To avoid overtraining the model, we used an early stopping technique to stop training the model if, after five consecutive iterations, the model did not get higher accuracy than previous iterations or if the loss value got higher than 1. At this stage, if the model accuracy was still not high enough, we restructured the model layers by changing the size of the sliding window, pooling window, and the number of layers.

Figure 9 shows the values of validation accuracy and loss of *CoLA* per epoch. We trained the model for 1500 epochs with the primary dataset, which is 240 elements of data without augmentation, 4560 elements of data with Keras-only augmentation, and 10560 elements of data with all the augmentation techniques mentioned above. As illustrated, with a small amount of data, the accuracy cannot go higher than 69%, and the loss stays at a high rate of 4% which both are not ideal. On the other hand, if we feed enough data to the model, we can gain an accuracy of 97.3% for the validation phase with a loss value of around 0.05%, two of which show the model's proper functionality. The validation accuracy ensures that, unlike GNN models, *CoLA* learns features beyond the structure and topology of the circuit.

As a neural network uses inputs, weights, and activations to predict the label, the values of each of the numbers affect the model's accuracy. The distribution of the values of weights and activations is represented in Fig. 10. As the figure shows, over 99% of the numbers fall within the 8-bit representation range, and less than 1% of the numbers place in the 16-bit range. Consequently, using an 8-bit quantized model, we can still gain the same accuracy as the original model gives us. Table 6 shows a group of data to compare the execution time of the quantized model and the original

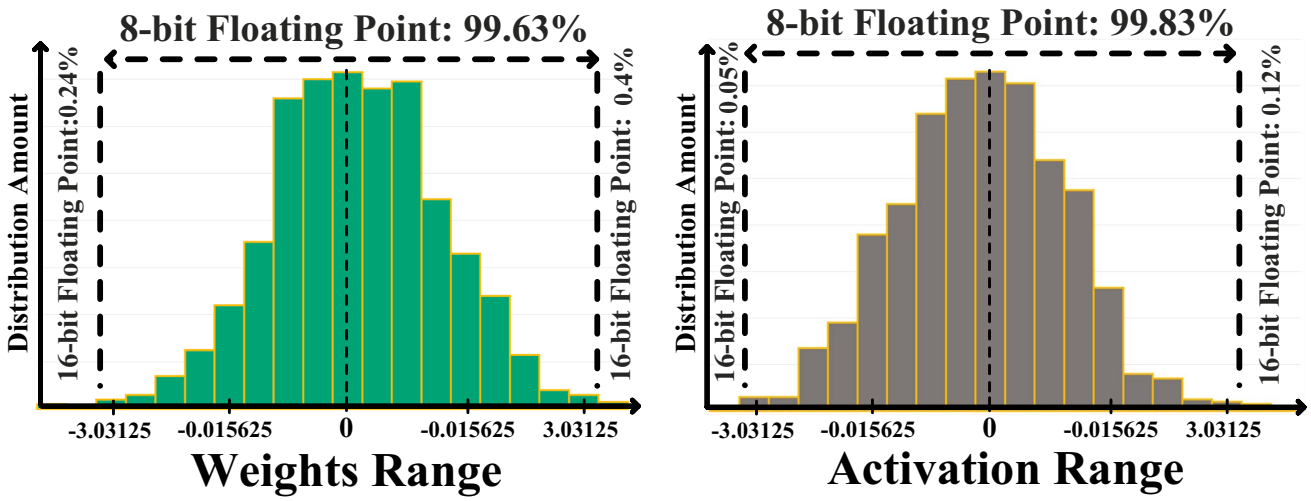


Fig. 10 *CoLA*: Distribution of the weights and activation values

Table 6 *CoLA*: Label prediction and execution time on a group of benchmarks on the validation dataset using the 8-bit quantized model. The augmentation type is resynthesis. Prediction LSL, prediction Label with quantized model; Q time, quantized model execution time; R time, regular model execution time

Benchmarks	Overhead	Q time (ms)	R time (ms)	Prediction LSL	Same label?
ex1010	5%	360	1179	Anti-SAT	Yes
ex1010	10%	173	612	Anti-SAT	Yes
c3540	25%	271	843	Anti-SAT	Yes
c7552	5%	149	577	Anti-SAT	No
c7552	5%	159	593	Anti-SAT	Yes
c1355	5%	124	541	SAR-Lock	No
c1355	10%	169	627	SAR-Lock	Yes
c3450	5%	173	663	R &D-C	Yes
c3540	10%	233	760	R &D-C	Yes
c7552	10%	207	827	R &D-C	Yes
ex1010	25%	268	873	BLE	Yes
c2670	5%	145	659	BLE	Yes
c6288	5%	142	736	BLE	Yes
c7552	25%	186	619	BLE	Yes

model. We used a validation set with some benchmark sizes larger than the training dataset to check the accuracy of the model for the larger, unseen data. Results show that the quantized *CoLA* assigns the logic locking faster than the original model with a negligible loss. The accuracy of the quantized model is 95.61% on 1056 items of validation data, whereas the original model is 97.3%.

5.3 *MADELINE* and *CoLA* Validation on Unseen Locking

MADELINE and *CoLA* work well on the datasets that are familiar to the model, with accuracy values of 99.01% and 97.3%, respectively. However, to study the models' efficiency, we should test the models' performance with circuits locked with a brand new method. We use UNSAIL [48] for this purpose. Feeding UNSAIL benchmarks to *MADELINE* does not require any additional steps since its primary goal is label prediction, but because *CoLA* assigns a locking method to unlocked benchmarks, it needs some modification to work with locked data properly. In this regard, we manipulated *CoLA* to assign a locking method only if the input benchmark needs it; otherwise, it returns nothing, which means the circuit is locked securely. In addition, to prepare an adequate amount of data for prediction, we utilized data augmentation and circuit resynthesis. A set of publicly available UNSAIL benchmarks is shown in table 7. The terms "v1" to "v4" refer to different versions of UNSAIL locking, which leads to various locked structures and a different number of gates.

Table 8 shows average accuracy for each of the UNSAIL benchmarks using *MADELINE* and *CoLA*. As we can see in the results of this table, *MADELINE* accuracy drops dramatically because circuit data in text format provides details about the number of each gate in the circuit but does not give enough information on the entire structure of the circuit as well as its functionality. On the other hand, *CoLA*'s accuracy ranges from 72.06% to 88.75% with an average of 80.11% which is still pretty much acceptable given the fact that UNSAIL locking was completely new to the model. *CoLA* works better than *MADELINE* because it gets the data in the shape of images, so it has the chance to distinguish different resynthesized versions of the same benchmarks and learn benchmarks' features beyond the structure of the circuit and finds information about its functionality. However, we cannot determinably tell if this is always the case for other unseen locking methods. One note to mention here is that training is a one-time offline phase, and the models can be re-trained at any time on new locking schemes.

Table 7 Number of gates in UNSAIL dataset with #Keysize=128

Benchmarks	c880	c1908	c2670	c3540	c5315	c6288	c7552
v1	347	360	641	1063	1338	2438	1377
v2	509	-	636	1066	1352	2440	1337
v3	401	-	633	1071	1337	2511	1385
v4	398	-	638	1078	1329	-	1372

Table 8 Average label prediction for UNSAIL benchmarks using *MADELINE* and *CoLA*

Benchmarks	c880	c1908	c2670	c3540	c5315	c6288	c7552	Average
MADELINE	60.72%	59.24%	56.03%	53.48%	52.66%	50.12%	50.01%	**54.60%**
CoLA	88.75%	86.40%	83.37%	80.92%	76.46%	72.81%	72.06%	**80.11%**

Table 9 Average label prediction for CAC benchmarks with #Keysize=64 using *MADELINE* and *CoLA*

Benchmarks	b14_C V1	b14_C V2	b14_C V3	b14_C V4	b14_C V5	Average
MADELINE	51.23%	55.24%	56.02%	51.93%	54.31%	**53.75%**
CoLA	80.24%	82.07%	78.49%	77.96%	79.22%	**79.20%**

5.4 *MADELINE* and *CoLA* Validation on New Benchmark and Unseen Locking

As another experiment, we study the models' performance with new benchmarks locked with an unseen locking method. We use 5 different synthesized versions of b14_C [69] locked with CAC method [19] with a key size of 64. The benchmarks are available in the repository of Valkyrie [52]. We converted .V files to .BENCH in order to feed them to the models. Table 9 shows average accuracy using *MADELINE* and *CoLA*. The results are almost the same as in Section 5.3 where we had a new locking method but with the benchmark that has been used in the training phase. This may lead to the conclusion that our models can perform the same on new and unseen circuits, which is the ultimate goal of the paper.

6 Conclusion

Because of the rising threat of diverse attacks on logic locking, evaluating the security and overhead of logic-locked digital circuits is more critical than ever. In this paper, to evaluate the security of logic locking methods, we proposed *MADELINE*, a DT-based model that receives circuits in the form of text data and uses the error rate to report whether or not a locking method is "safe" for a specific circuit. Then, we proposed *CoLA*, a CNN-based model that receives circuits in the form of image data and employs key correctness values and area overhead to assign a low-overhead and secure locking method to a given circuit. Experimental results showed that although both models received high accuracy in the case of unseen benchmarks, in the case of unseen locking methods, *MADELINE* performs poorly while *CoLA* still keeps up a reasonable accuracy.

We do think that there is no foolproof approach to preventing a zero-day attack, but there is immense value in having a reliable and proactive framework to assess the overhead and security of a newly proposed logic locking technique. For future works, the explainability of the decisions made by the ML models can be pursued to identify secure yet low-overhead structures in digital circuits for a design-for-security approach.

Statements and Declarations

Funding This material is based upon work supported by the National Science Foundation under Award No. 2245247.

Competing Interests The authors declare no competing interests.

Author Contributions YA and AR contributed equally to this work.

Data Availibility The dataset is available from the corresponding author upon request.

Ethical Approval Not applicable

References

1. Roy JA,Koushanfar F, Markov IL (2008) Epic: Ending piracy of integrated circuits. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1069-1074
2. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2013) Fault analysis-based logic encryption. In: IEEE Transactions on Computers, pp 410-424
3. Dupuis S, Ba PS, Natale GD, Flottes ML, Rouzeyre B (2014) A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In: International On-Line Testing Symposium (IOLTS), pp 49-54
4. Baumgarten A, Tyagi A, Zambreno J (2010) Preventing IC piracy using reconfigurable logic barriers. In: IEEE design & Test of computers, pp 66-75
5. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic locking algorithms In International Symposium on Hardware Oriented Security and Trust (HOST), pp 137-143
6. Shamsi K, Li M, Meade T, Zhao Z, Pan DZ, Jin Y (2017) AppSAT: approximately deobfuscating integrated circuits. In: International Symposium on Hardware Oriented Security and Trust (HOST), pp 95-100
7. Shen Y, Zhou H (2017) Double DIP: re-evaluating security of logic encryption algorithms. In: Great Lakes Symposium on VLSI (GLSVLSI), pp 179-184
8. Rezaei A, Shen Y, Zhou H (2020) Rescuing logic encryption in post-SAT era by locking & obfuscation. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 13-18

9. Plaza SM, Markov IL (2015) Solving the third-shift problem in IC piracy with test-aware logic locking. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp 961-971
10. Rezaei A, Afsharmazayejani R, Maynard J (2022) Evaluating the security of eFPGA-based redaction algorithms. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 154:1-7
11. Zhou H, Jiang R, Kong S (2017) CycSAT: SAT-based attack on cyclic logic encryptions. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 49-56
12. Shen Y, Li Y, Rezaei A, Kong S, Dlott D, Zhou H (2019) BeSAT: behavioral SAT-based attack on cyclic logic encryption. In: Proceedings of the 24th Asia and South Pacific Design Automation Conference, pp 657-662
13. Shen Y, Li Y, Kong S, Rezaei A, Zhou H (2019) SigAttack: new high-level SAT-based attack on logic encryptions. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 940-943
14. Shamsi K, Pan DZ, Jin Y (2019) IcySAT: improved SAT-based attacks on cyclic locked circuits. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 1-7
15. McDaniel I, Zuzak M, Srivastava A (2022) A black-box sensitization attack on SAT-hard instances in logic obfuscation. In: IEEE International Conference on Computer Design (ICCD), pp 239-246
16. Yasin M, Mazumdar B, Rajendran J, Sinanoglu O (2016) SARLock: SAT attack resistant logic locking. In: International Symposium on Hardware Oriented Security and Trust (HOST), pp 236-241
17. Xie Y, Srivastava A (2019) Anti-SAT: mitigating SAT attack on logic locking. IEEE Trans Comput-Aided Des Integr Circuits Syst 38(2):199–207
18. Yasin M, Sengupta A, Nabeel MT, Ashraf M, Rajendran J, Sinanoglu O (2017) Provably-secure logic locking: from theory to practice. In: ACM SIGSAC Conference on Computer and Communications Security (CCS), pp 1601-1618
19. Shamsi K, Meade T, Li M, Pan DZ, Jin Y (2019) On the approximation resiliency of logic locking and IC camouflaging schemes. IEEE Trans Inf Forensics Secur 14(2):347–359
20. Shamsi K, Li M, Meade T, Zhao Z, Pan DZ, Jin Y (2017) Cyclic obfuscation for creating SAT-unresolvable circuits. In: Proceedings of the on Great Lakes Symposium on VLSI, pp 173-178
21. Rezaei A, Shen Y, Kong S, Gu J, Zhou H (2018) Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 85-90
22. Rezaei A, Li Y, Shen Y, Kong S, Zhou H (2019) CycSAT-unresolvable cyclic logic encryption using unreachable states. In: Asia and South Pacific Design Automation Conference (ASP-DAC), pp 358-363
23. Rezaei A, Zhou H (2021) Sequential logic encryption against model checking attack. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1178-1181
24. Hu B, Tian J, Shihab M, Reddy G, Swartz W, Makris Y, Schaefer BC, Sechen C (2019) Functional obfuscation of hardware accelerators through selective partial design extraction onto an embedded FPGA. In: Great Lakes Symposium on VLSI (GLSVLSI), pp 171-176
25. Mohan P, Atli O, Sweeney J, Kibar O, Pileggi L, Mai K (2021) Hardware redaction via designer-directed fine-grained eFPGA insertion. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1186-1191
26. Bhandari J, Moosa A, Tan B, Pilato C, Gore G, Tang X, Temple S, Gaillardon P, Karri R (2021) Exploring eFPGA-based redaction for IP protection. In: International Conference On Computer Aided Design (ICCAD), pp 1-9
27. Roshanisefat S, Kamali HM, Homayoun H, Sasan A (2020) SAT-hard cyclic logic obfuscation for protecting the IP in the manufacturing supply chain. In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp 954-967
28. Kamali HM, Azar KZ, Gaj K, Homayoun H, Sasan A (2018) LUT-Lock: a novel LUT-based logic obfuscation for FPGA bitstream and ASIC-hardware protection. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 405-410
29. Zhang D, He M, Wang X, Tehranipoor M (2017) Dynamically obfuscated scan for protecting IPs against scan-based attacks throughout supply chain. In: VLSI Test Symposium (VTS), pp 1-6
30. Karmakar R, Kumar H, Chattopadhyay S (2019) Efficient key gate placement and dynamic scan obfuscation towards robust logic encryption. In: IEEE Transactions on Emerging Topics in Computing
31. Zhou H, Rezaei A, Shen Y (2019) Resolving the trilemma in Logic encryption. In: IEEE International Conference on Computer Aided Design (ICCAD), pp 1-8
32. Afsharmazayejani R, Sayadi H, Rezaei A (2022) Distributed logic encryption: essential security requirements and low-overhead implementation. In: Proceedings of Great Lakes Symposium on VLSI (GLSVLSI), pp. 127-131
33. Rezaei A, Hedayatipour A, Sayadi H, Aliasgari M, Zhou H (2022) Global attack and remedy on IC-specific logic encryption. In: IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 145-148
34. Zhang Y, Hu Y, Nuzzo P, Beerel PA (2022) "TriLock: IC protection with tunable corruptibility and resilience to SAT and removal attacks. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1329-1334
35. Maynard J, Rezaei A (2023) DK lock: dual key logic locking against oracle-guided attacks. In: International Symposium on Quality Electronic Design (ISQED), pp. 1-7
36. Sisejkovic D, Reimann LM, Moussavi E, Merchant F, Leupers R (2021) Logic locking at the frontiers of machine learning: a survey on developments and opportunities. In: IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 1-6
37. Chakraborty P, Cruz J, Bhunia S (2018) SAIL: machine learning guided structural analysis attack on hardware obfuscation. In: Asian Hardware Oriented Security and Trust Symposium (Asian-HOST), pp. 56-61
38. Chen H, Fu C, Zhao J, Koushanfar F (2019) GenUnlock: an automated genetic algorithm framework for unlocking logic encryption. In: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1-8
39. Alrahis L, Patnaik S, Hanif MA, Saleh H, Shafique M, Sinanoglu O (2021) GNNUnlock+: a systematic methodology for designing graph neural networks-based oracle-less unlocking schemes for provably secure logic locking. IEEE Trans Emerg Topics Comput 10(3):1575-1592
40. Chen H, Fu C, Zhao J, Koushanfar F (2022) GALU: a genetic algorithm framework for logic unlocking. Research and Practice, In Digital Threats
41. Azar KZ, Kamali HM, Homayoun H, Sasan A (2020) NNgSAT: neural network guided SAT attack on logic locked complex structures. In: IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp 1-9
42. Shamsi K, Zhao G (2022) An oracle-less machine-learning attack against lookup-table-based logic locking. In: Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI), pp. 133-137
43. Alrahis L, Patnaik S, Hanif MA, Shafique M, Sinanoglu O (2021) UNTANGLE: unlocking routing and logic obfuscation using graph neural networks-based link prediction. In: IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1-9
44. Alrahis L, Patnaik S, Shafique M, Sinanoglu O (2021) OMLA: an oracle-less machine learning-based attack on logic locking. IEEE Trans Circuits Syst II Express Briefs 69(3):1602–1606

45. Sisejkovic D, Merchant F, Reimann LM, Srivastava H, Hallawa A, Leupers R (2021) Challenging the security of logic locking schemes in the era of deep learning: a neuroevolutionary approach. J Emerg Technol Comput Syst 17(3):30
46. Arp D, Quiring E, Pendlebury F, Warnecke A, Pierazzi F, Wressnegger C, Cavallaro L, Rieck K (2022) Dos and don'ts of machine learning in computer security. In: Proceedings of USENIX Security Symposium
47. Pilato C, Chowdhury AB, Sciuto D, Garg S, Karri R (2021) ASSURE: RTL locking against an untrusted foundry. IEEE Trans Very Large Scale Integr (VLSI) Syst 29(7):1306-1318
48. Alrahis L, Patnaik S, Knechtel J, Saleh H, Mohammad B, Al-Qutayri M, Sinanoglu O (2021) UNSAIL: thwarting oracle-less machine learning attacks on logic locking. IEEE Trans Inf Forensics Secur 16:2508–2523
49. Becker GT, Regazzoni F, Paar C, Burleson WP (2013) Stealthy dopant-level hardware Trojans. In: International Workshop on Cryptographic Hardware and Embedded Systems (CHES), pp 197-214
50. Alasad Q, Yuan J (2017) Logic obfuscation against IC reverse engineering attacks using PLGs. In: IEEE International Conference on Computer Design (ICCD), pp 341-344
51. Rezaei A, Gu J, Zhou H (2019) Hybrid memristor-CMOS obfuscation against untrusted foundries. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp 535-540
52. Limaye N, Patnaik S, Sinanoglu O (2022) Valkyrie: vulnerability assessment tool and attack for provably-secure logic locking techniques. IEEE Trans Inf Forensics Secur 17:744–759
53. Elnaggar R, Chakrabarty K (2018) Machine learning for hardware security: opportunities and risks. In: Journal of Electronic Testing, pp 183-201
54. Liu W, Chang CH, Wang X, Liu C, Fung JM, Ebrahimabadi M, Karimi N, Meng X, Basu K (2021) Two sides of the same coin: boons and banes of machine learning in hardware security. In: IEEE Journal on Emerging and Selected Topics in Circuits and Systems, pp 228-251
55. Tan B, Karri R (2020) Challenges and new directions for AI and hardware security. In: IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pp 277-280
56. Darjani A, Kavand N, Rai S, Kumar A (2023) Discerning limitations of GNN-based attacks on logic locking. In: Design Automation Conference (DAC), pp 1-6
57. Li M, Khan S, Shi Z, Wang N, Yu H, Xu Q (2022) DeepGate: learning neural representations of logic gates. In: ACM/IEEE Design Automation Conference (DAC), pp 667-672
58. Wang Z, Bai C, He Z, Zhang G, Xu Q, Ho T-Y, Yu B, Huang Y (2022) Functionality matters in netlist representation learning. In: ACM/IEEE Design Automation Conference (DAC), pp 61-66
59. Yasaei R, Yu S-Y, Naeini EK, Faruque MAA (2021) GNN4IP: graph neural network for hardware intellectual property piracy detection. In: ACM/IEEE Design Automation Conference (DAC), pp 217-222
60. Chowdhury AB, Bhandari J, Collini L, Karri R, Tan B, Garg S (2023) ConVERTS: contrastively learning structurally invariant netlist representations. In: ACM/IEEE Workshop on Machine Learning for CAD (MLCAD), pp 1-6
61. Navada A, Ansari AN, Patil S, Sonkamble BA (2011) Overview of use of decision tree algorithms in machine learning. In: IEEE control and system graduate research colloquium, pp 37-42
62. Quinlan JR (1986) Induction of decision trees. In: Machine learning, pp 81-106
63. Berkeley Logic Synthesis and Verification Group. ABC: a system for sequential synthesis and verification. http://www.eecs.berkeley.edu/alanmi/abc/
64. Keras: deep learning for humans. http://github.com/fchollet/keras/
65. Khosla C, Saini BS (2020) Enhancing performance of deep learning models with different data augmentation techniques: a survey. In: International Conference on Intelligent Engineering and Management (ICIEM), pp. 79-85
66. Brglez F, Fujiwara H (1985) A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 677-692
67. Yang S (1991) Logic synthesis and optimization benchmarks user guide version 3.0. In: Microelectronics Center of North Carolina (MCNC) International Workshop on Logic Synthesis
68. NEOS: netlist encryption and obfuscation suite. http://bitbucket.org/kavehshm/neos/
69. Davidson S (1999) ITC'99 benchmark circuits - preliminary results. In: International Test Conference (ITC), pp 1125-1125