CoLA: Convolutional Neural Network Model for Secure Low Overhead Logic Locking Assignment

Yeganeh Aghamohammadi
University of California, Santa Barbara
yeganeh@ucsb.edu

Amin Rezaei
California State University, Long Beach
amin.rezaei@csulb.edu

ABSTRACT

Chip designers can secure their ICs against piracy and overproduction by employing logic locking and obfuscation. However, there are numerous attacks that can examine the logic-locked netlist with the assistance of an activated IC and extract the correct key using a SAT solver. In addition, when it comes to fabrication, the imposed area overhead is a challenge that needs careful attention to preserve the design goals. Thus, to assign a logic locking method that can provide security against diverse attacks and at the same time add minimal area overhead, a comprehensive understanding of the circuit structure is needed. Towards this goal, in this paper, we first build a multi-label dataset by running different attacks on benchmarks locked with existing logic locking methods and various key sizes to capture the provided level of security and overhead for each benchmark. Then we propose and analyze *CoLA*, a convolutional neural network model that is trained on this dataset and thus is able to map circuits to secure low-overhead locking schemes by analyzing extracted features of the benchmark circuits. Considering various resynthesized versions of the same circuits empowers *CoLA* to learn features beyond the structure view alone. We use a quantization method that can lower the computation overhead of feature extraction in the classification of new, unseen data, hence speeding up the locking assignment process. Results on over 10,000 data show high accuracy both in the training and validation phases.

CCS CONCEPTS

• **Security and privacy** → **Security in hardware**; • **Computing methodologies** → *Supervised learning*; • **Hardware** → Hardware reliability screening.

KEYWORDS

Logic Locking; Obfuscation; SAT Attack; Machine Learning; Convolutional Neural Network; Low-Overhead

ACM Reference Format:
Yeganeh Aghamohammadi and Amin Rezaei. 2023. CoLA: Convolutional Neural Network Model for Secure Low Overhead Logic Locking Assignment. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590219

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590219

1 INTRODUCTION

As fabless manufacturing is at a crucial point, it is momentous that semiconductor and electronic industries protect their hardware Intellectual Properties (IPs) from malicious threats. Due to the high cost of the hardware design, the designers need to preserve their Integrated Circuits (ICs) from piracy. In addition, unauthorized overproduction by untrusted third-party foundries puts IC designers at risk of not receiving the expected revenue for their products. One of the promising solutions against these malicious threats is logic locking and obfuscation [1–3], which adds extra gates controlled by secret key inputs to each IC. In this case, the circuit works properly only when the correct key is inserted, and it malfunctions otherwise.

While there are more and more sequences of defenses [4–11] and attacks [12–20] in logic locking area, the job of the defender becomes more effortful because if a logic locking method is secure against all but one attack, the defender is doomed! Thus, from the defense perspective, one question becomes more critical than before: *how secure is a logic-locked circuit?* On the other hand, the main goal of logic locking is to protect an efficient and precious design from being pirated and overproduced, and it is counter-intuitive if the locking overhead is beyond a small, reasonable amount. So, another important question is: *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 least amount of 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. While all of these initiatives are on the attacker side [21–23], to the best of our knowledge, no research has been conducted on the opportunities that ML-based analysis might provide for hardware designers to securely and yet inexpensively protect their ICs in a fabless paradigm. We believe that ML can make hardware protection easier, more proactive, and more affordable. However, it can only do so if the underlying data gives a comprehensive view of the environment. The Convolutional Neural Network (CNN) is one of the best candidates to extensively extract features of different circuits. Because the CNNs' inputs are in the shape of images and there is no available image dataset for training, we create an image dataset of the locked benchmarks. With the help of data augmentation, the CNN model can be trained on thousands of data, gain high accuracy, and become noise-resilient. The main contributions of this work are as follows:

- Developing a multi-label security and overhead degree dataset consisting of more than 10,000 benchmarks locked with 8 distinct logic locking methods;
- Building and training an accurate CNN-based ML model under 6 different attacks with hyperparameter tuning;

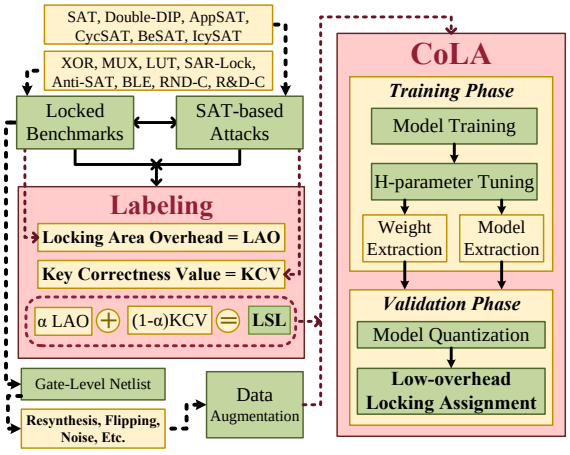

Figure 1: CoLA framework

- Testing the created CNN model on seen and unseen logic-locked benchmarks and evaluating the model's security and overhead prediction accuracy, which is greater than 97% in the original model and greater than 95% in the quantized version.

2 BACKGROUND AND RELATED WORKS

The selection of logic locking techniques in our machine learning model is based on the manifold of the methods, consisting of traditional pre-SAT approaches [1–3] and advanced SAT-resilient [4, 5], compound [6] and cyclic methods [7, 8]. The choice of logic locking attacks is commonly based on the most powerful SAT-based oracle-guided attacks whose repositories are publicly available [12–17]. On the other hand, the choice of the machine learning method is centered on its functionality, data format, and prediction time. However, the training process of our universal model is not dependent on the selection of defenses and attacks, and other locking/unlocking methods can be applied with the same procedure to further train the model.

2.1 Logic Locking Defenses

We can divide logic locking methods into pre-SAT and post-SAT categories. In pre-SAT XOR-based locking [1], a combination of random buffers and inverters get matched with the key bits. Afterwards, with the usage of key bit controlled XOR gates, it replaces the selected buffers and inverters. If the hiding gate is a buffer, the correct key bit is "0" while if that is 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 LUT-based locking [3] 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 use of LUTs ensures the existence of such barriers. The key inputs are the values stored in the lookup tables.

Both SAR-Lock [4] and Anti-SAT [5] are post-SAT locking solutions that prevent the correct key leakage to 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-based attack [12] will take exponentially more iterations to find the correct key with respect to the key size. As an advanced post-SAT method, Bilateral Logic Encryption (BLE) [6] 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. In [7], random cycle insertion (RND-Cycle) is presented, which inserts dummy 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. Last but not least, dummy and real cycle insertion (R&D-cycle) [8] turns an acyclic combinational circuit to a cyclic circuit before cyclically locking it. In all of the above logic locking methods, the correct key must be put in a tamper-proof memory soon after the fabricated ICs return from the foundry.

2.2 Logic Locking Attacks

The original SAT-based attack [12] finds unique input patterns called DIPs to exclude equivalence classes of keys.

To attack post-SAT locking approaches like SAR-Lock [4] and Anti-SAT [5], Double-DIP [13] 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 time consumption, which is not ideal for real-world problems. AppSAT [14] 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 [7], an oracle-guided attack called CycSAT [15] is proposed which assumes 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 the possible cycles in a cyclically locked circuit, BeSAT [16] pursues a behavioral method to unlock cyclic logic locking with the goal of reducing the missing cycle problem of CycSAT. Finally, IcySAT [17] has been introduced that follows a cycle unrolling approach, contrary to CycSAT which follows a cycle breaking strategy.

2.3 Machine Learning Models

One of the machine learning methods to interpret the circuit structure is Graph Neural Network (GNN) which comes in handy. To extract features of ICs, a GNN uses an undirected graph to show the netlist, representation of circuits, and the gates' connectivity. In [24], an end-to-end GNN framework is proposed to predict the runtime of the original SAT-based attack with respect to the extracted features of locked benchmarks using an adjacency matrix. Although, graph representation preserves the topology of the circuit, using an undirected graph for netlist representation is one of

the shortcomings of GNNs because the inputs/outputs neighborhood of the netlist will be indistinguishable. Furthermore, existing GNN models perform poorly in the presence of noise and changes in the circuit structure, implying that the model is highly dependent on the specific gates in the circuit but not on their dependent functionalities to each other. For instance, in [22] authors incorporates a standard cell library to assign gate feature maps to the GNN model, which makes the model strictly dependent on the structure of the circuit. Thereby, we use a regular CNN to receive image data of the circuits with various structures as input. With the help of resynthesis and data augmentation techniques, we can provide more circuit structures for the same function to create a large-sized dataset. This helps the model predict the label of new, unseen circuits with high accuracy. In addition, the model will be able to recognize different circuit structures under the same functionality.

3 CONVOLUTIONAL NEURAL NETWORK LOGIC LOCKING ASSIGNMENT MODEL

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 circuit. First, we gather data and use the area overhead and the error rate 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. The *CoLA* framework is shown in Fig. 1.

3.1 Data Gathering, Labeling, and Augmentation

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 security 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.

Error Rate (ER) & Key Correctness Value (KCV): We define ER of a key as the number of input patterns in which the locked circuit under that key and the oracle are the same, divided by all the input patterns. ER equal to 0 means that the key found by an attack is the exact correct key. 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 point of view, the higher the ER (i.e., the lower the KCV), the more secure the locking technique.

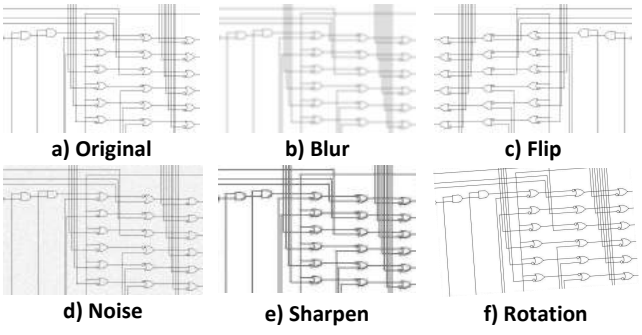

Figure 2: Visual examples of data augmentation

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 area and error rate, we define Low-overhead & Secure Label (LSL) as the following:

$$LSL = \alpha LAO + (1 - \alpha)KCV \quad (1)$$

where α is the weighted coefficient with an amount between 0 and 1 to put emphasis on the key correctness side or the area side. Based on Equation 1, 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 lowest KCV (highest security) and the lowest LAO (lowest area). As a circuit has multiple key ERs because of the usage of different attack methods, we use the lowest key ER for Equation 1 to account for the highest vulnerability of a locking method against any of the attacks.

While thousands of data are needed to efficiently train a CNN, 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 [25]. Then, using different routing settings, we export multiple random resynthesized layouts for each benchmark in terms of the structure of the layouts, and the positions of the elements. The different layouts capture variations in logic simplification, grouping of related nodes, showing registers without fan-outs, and enabling global net routing. As shown in Fig. 2, to get even more data, we apply data augmentation techniques available in *Keras* [26] such as noise injection, random brightness, random flip, and rotating, to name a few [27]. 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.

3.2 Training CoLA

CoLA is a CNN-based machine learning model aimed at assigning a low-overhead locking technique to unseen circuits. 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

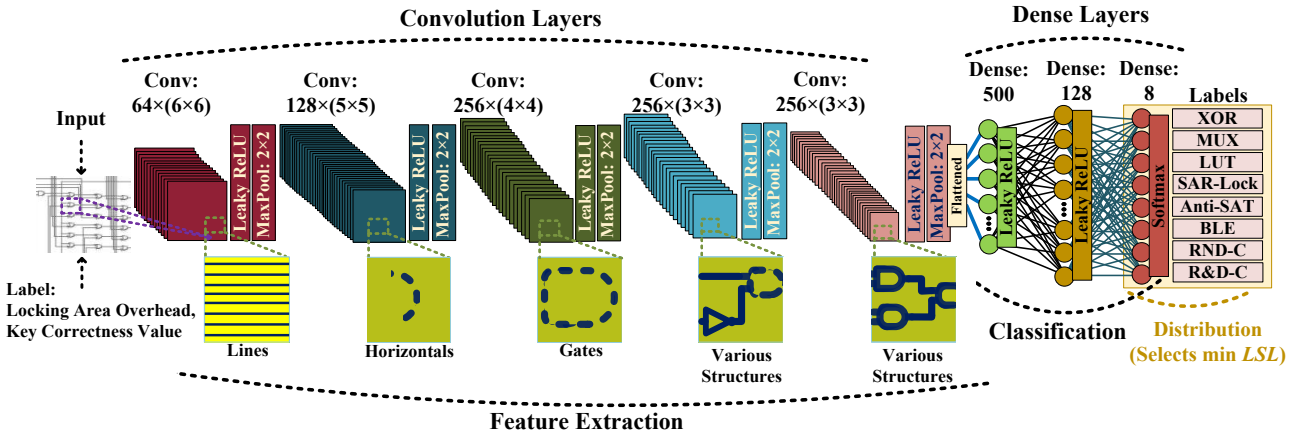

Figure 3: CoLA structure

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 unseen data as well. 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.

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 followed by 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. 3.

3.3 Evaluating CoLA

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. If the majority of the numbers place in the range we demand, we can get high accuracy of the model, while consuming fewer resources. A quantized model uses less memory and processes the computation faster. In the case of overflow and underflow in the quantization process, we assign the highest and lowest range demand, respectively.

4 EXPERIMENTAL RESULTS

We created *CoLA* on an Intel Core i7-10750H CPU, with a RAM size of 16 GB. To build CNN, we used *Python* and the *Tensorflow* package. To extract data in the form of images, we used the web edition of Intel Quartus II. To create a labeled dataset, we gathered data using the circuit benchmarks of MCNC'91 [28] and ISCAS'85 [29], and different locking methods [1–8] on each benchmark shown in Table 1. We recorded the LAOs by comparing the area of each locked benchmark with its original version. To gather the ERs, we ran different attacks [12–17] on the locked benchmarks and chose the minimum ER among the reported keys for each benchmark. The setup for SAT [12], Double-DIP [13], and CycSAT [15] attacks is the same as the default setup, and we consider it a timeout with an error rate of 1, if the key cannot be found in one day of running. For all the other attack methods (i.e., AppSAT [14], BeSAT [16], and IcySAT [17]) we used the *NEOS* suite [30] with a default setup. Then, we used equation 1 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 LAO and ER in our dataset is shown in Fig. 4. Finally, we converted the benchmarks to image netlists and augmented the

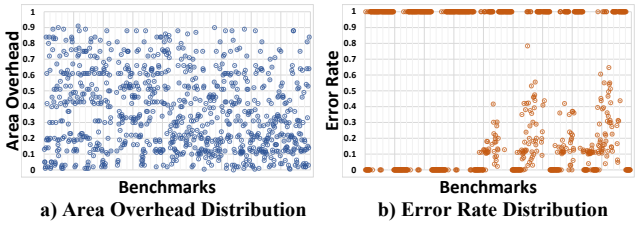

Figure 4: Area overhead and error rate distributions of the dataset

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

Bench	#In	XOR-based [1]		MUX-based [2]		LUT-based [3]		SAR-Lock [4]		Anti-SAT [5]		BLE [6]		RND-Cycle [7]		R&D-Cycle [8]	
		#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K	#G	#K
apex2	39	643	31	642	32	1780	292	644	31	687	38	713	40	630	20	670	40
apex4	10	5628	268	5628	269	18441	3356	5633	268	5381	10	5388	10	5380	20	5420	40
c432	36	170	8	170	10	1101	184	168	8	233	36	254	36	180	20	220	40
c499	41	212	10	214	12	1891	288	212	10	291	44	310	42	222	20	262	40
c880	60	404	19	385	22	1831	3112	403	19	504	60	536	60	403	20	443	40
c1908	33	925	44	734	46	1586	200	928	44	945	32	968	34	900	20	940	40
c5315	178	2427	115	2113	124	7841	1176	2424	115	2307	134	1495	178	2327	20	2367	40
dalu	75	2418	115	2417	119	4875	640	2436	115	2447	74	2491	76	2318	20	2358	40
des	256	6804	324	6809	336	17879	2856	6804	324	6550	38	7116	256	6493	20	6533	40
ex5	8	1109	53	1108	53	4627	888	1109	53	1072	8	1078	8	1075	20	1115	40
i4	192	360	17	365	27	1538	272	355	17	527	94	821	192	358	20	398	40
i7	199	1384	66	1391	76	4921	908	1389	66	1340	12	1818	200	1335	20	1375	40
i8	133	2589	123	2594	130	8144	1348	2598	66	2533	34	2802	134	2484	20	2524	40
i9	88	1089	52	1091	56	3477	608	1092	52	1088	26	1258	88	1055	20	1095	40
k2	46	1908	91	1907	93	4482	620	1906	91	1908	46	1933	46	1835	20	1875	40
seq	41	3697	176	3697	178	10829	1848	3700	176	3600	40	3627	42	3539	20	3579	40

gathered data to more than 10,000 samples using the approaches discussed in Section 3.

After creating the dataset, we trained *CoLA* and extracted the features. Then, a quantized version of *CoLA* is used to get validation computation fast. The accuracy of the training phase and testing phase can be affected by several hyperparameters. Using *Keras* [26] hyperparameter tuning, we increased the accuracy of the training and validation phases with a loss value of below 0.1% and without overfitting and 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.

Fig. 5 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

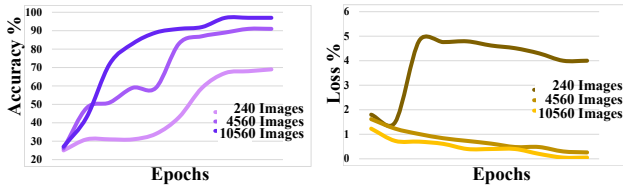

Figure 5: Validation accuracy and loss of *CoLA* per epoch

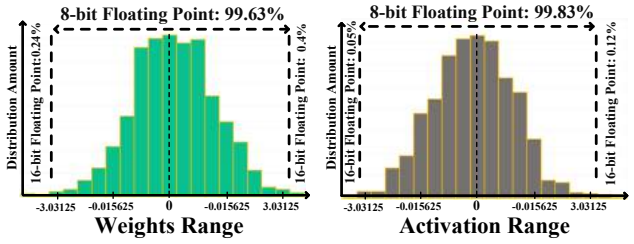

Figure 6: Distribution of the weights and activation values

elements of data with all the augmentation techniques mentioned. 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 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. 6. 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 achieve a very high level of accuracy. Table 2 shows a group of data to compare the execution time of the quantized model and the original model. We used a validation

Table 2: Label prediction and execution time on a group of benchmarks on the validation dataset using the 8-bit quantized *CoLA*. The augmentation type is resynthesis. Pred. LSL: Prediction Label with quantized *CoLA*, Q time: Quantized model execution time, R time: Regular model execution time.

Bench	Overhead	Q time (ms)	R time (ms)	Pred. 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

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 CONCLUSION

In this paper, we explored logic locking and obfuscation from a defense point of view to answer two important questions: *1) how secure is a logic-locked circuit?*, and *2) how much overhead is imposed by locking?* First, we gathered several benchmarks locked with different methods and assigned a label to each, based on both their security against state-of-the-art attacks and their overhead compared to the original ones. Then, we augmented the gathered data to more than 10,000 image circuits by using resynthesis as well as image data augmentation methods. Finally, we proposed *CoLA*, a CNN-based low-overhead logic locking assignment model that extracts features from given data and uses them to assign a low-overhead lock to a given circuit using an 8-bit quantized model.

Experimental results show that not only *CoLA* can efficiently assign a secure low-overhead lock on new, unseen logic circuits, but it can also pave the way for a comparative model for the security and overhead analysis of future logic locking methods.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under Award No. 2245247.

REFERENCES

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1069-1074, 2008.

[2] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," In *IEEE Transactions on computers*, pp. 410-424, 2013.

[3] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," In *IEEE design & Test of computers*, pp. 66-75, 2010.

[4] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 236-241, 2016.

[5] Y. Xie and A. Srivastava, "Anti-SAT: Mitigating SAT attack on logic locking," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199-207, 2019.

[6] A. Rezaei, Y. Shen, and H. Zhou, "Rescuing logic encryption in post-SAT era by locking & obfuscation," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 13-18, 2020.

[7] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating SAT-unresolvable circuits," In *Proceedings of the on Great Lakes Symposium on VLSI*, pp. 173-178, 2017.

[8] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 85-90, 2018.

[9] A. Rezaei and H. Zhou, "Sequential logic encryption against model checking attack," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1178-1181, 2021.

[10] A. Rezaei, J. Gu, and H. Zhou, "Hybrid memristor-CMOS obfuscation against untrusted foundries," In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 535-540, 2019.

[11] R. Afsharmazayejani, H. Sayadi, and A. Rezaei, "Distributed logic encryption: Essential security requirements and low-overhead implementation," In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 127-131, 2022.

[12] P. Subramanyan, S. Ray, S. Malik, "Evaluating the security of logic locking algorithms," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137-143, 2015.

[13] Y. Shen and H. Zhou, "Double DIP: Re-evaluating security of logic encryption algorithms," In *Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 179-184, 2017.

[14] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 95-100, 2017.

[15] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 49–56, 2017.

[16] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based attack on cyclic logic encryption," In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 657-662, 2019.

[17] K. Shamsi, D. Z. Pan, Y. Jin, " IcySAT: Improved SAT-based attacks on cyclic locked circuits," In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-7, 2019.

[18] Y. Shen, Y. Li, S. Kong, A. Rezaei, and H. Zhou, "SigAttack: New high-level SAT-based attack on logic encryptions," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 940-943, 2019.

[19] A. Rezaei, A. Hedayatipour, H. Sayadi, M. Aliasgari, and H. Zhou, "Global attack and remedy on IC-specific logic encryption," In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 145-148, 2022.

[20] A. Rezaei, R. Afsharmazayejani, and J. Maynard, "Evaluating the security of eFPGA-based redaction algorithms," In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, article 157, pp 1-7, 2022.

[21] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1-9, 2020.

[22] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "OMLA: An oracle-less machine learning-based attack on logic locking," In *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1602-1606, 2022.

[23] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," In *J. Emerg. Technol. Comput. Syst.* vol. 17, no. 3, article 30, 2021.

[24] Z. Chen, G. Kolhe, S. Rafatirad, C.T. Lu, S.M. PD, H. Homayoun, and L. Zhao, "Estimating the circuit de-obfuscation runtime based on graph deep learning," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 358-363, 2020.

[25] "ABC: A system for sequential synthesis and verification," http://eecs.berkeley.edu/~alanmi/abc/.

[26] "Keras: Deep learning for humans," http://github.com/fchollet/keras/.

[27] C. Khosla and B. S. Saini, "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, 2020.

[28] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," In *Microelectronics Center of North Carolina (MCNC) International Workshop on Logic Synthesis*, 1991.

[29] F. Brglez and H. Fujiwara, "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, 1985.

[30] "NEOS: Netlist encryption and obfuscation suite," http://bitbucket.org/kavehshm/neos/.