Logical Graph Deep Learning for Circuit Deobfuscation Runtime Estimation

Zhiqian Chen*, Gaurav Kolhe§, Setareh Rafatirad§, Chang-Tien Lu*,

Sai Manoj P.D.§, Houman Homayoun§, Liang Zhao§

Virginia Tech* George Mason University§

{czq, ctlu}@vt.edu, {gkolhe, srafatir, spudukot, hhomayou, lzhao9}@gmu.edu

ABSTRACT

Circuit obfuscation is a recently proposed defense mechanism to protect digital integrated circuits (ICs) from reverse engineering by logic locking. There have been effective schemes such as satisfiability-checking (SAT) based attacks that can potentially decrypt obfuscated circuits, called deobfuscation. Deobfuscation runtime could be days or years, depending on the layouts of the obfuscated ICs. And hence accurately pre-estimating the deobfuscation runtime within a reasonable amount of time is highly crucial for the defenders to optimize their defense. However, ICs are logical graphs whose edges represent logical operators and hence are radically different from existing works which typically focus graphs whose edges are for diffusion or message passing. To automatically learn the discriminative features and achieve accurate prediction, this work proposes the first framework that predicts the deobfuscation runtime based on graph deep learning. A conjunctive normal form (CNF) bipartite graph is formulated and leveraged to represent the original circuit in terms of the SAT hardness. To overcome the difficulty in capturing the dynamic size of the CNF graph, an energy-based kernel is proposed to aggregate dynamic features into an identical vector space. The proposed CNF-Net is an end-to-end framework which can automatically extract the determinant features for deobfuscation runtime. Extensive experiments demonstrate its effectiveness and efficiency.

1 INTRODUCTION

The considerable high capital costs on semiconductor manufacturing motivate most semiconductor companies to outsource their designed integrated circuits (ICs) to the contract foundries for fabrication. Despite the reduced cost and other benefits, this trend has led to ever-increasing security risks such as IC counterfeiting, piracy and unauthorized overproduction by the contract foundries [18]. The overall financial risk caused by such counterfeit and unauthorized ICs was estimated to be over \$169 billion per year [13]. The major threats from the attackers arise from reverse engineering an IC by fully identifying its functionality by stripping it layer-by-layer and extracting the unveiling gate-level netlist. To prevent such reverse engineering, IC *obfuscation* techniques have been extensively researched in recent years [25]. The general idea is to obfuscate some gates in an IC so that their gate type cannot be determined by reverse engineering optically, yet they preserve the functionality same as the original gates. As shown in Fig. (1), obfuscation is a process which selects a part of the circuit (in pink) and modifies the structure of them, whose functionality can be retrieved only if correct keys are provided at the input.

The runtime of state of the art attack [7, 11] to reverse engineer the IC mostly depends on the complexity of the obfuscated IC, which can vary from milliseconds to days or years. Therefore, a successful obfuscation requires attackers a prohibitive amount of time (i.e., many years) to deobfuscate. However, gates to obfuscate come at a heavy cost in finance, power, and space, such trade-off forces us to search for optimal layout instead of purely increasing their quantity. Therefore, the best set of gates for being obfuscated maximizes the runtime for deobfuscating. However, until now it is still generally based on human heuristics or experience, which is seriously arbitrary and suboptimal [10]. This is because it is unable to "try and error" all the different ways of obfuscation, as there are millions of combinations to try and the runtime for each try (i.e., to run the attacker) can be days, weeks, or years.

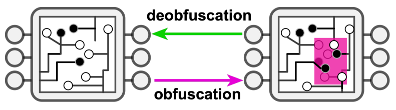


Figure 1: Illustration of obfuscation and deobfuscation

Our paper focuses on efficient and scalable ways to estimate the runtime for an attacker to deobfuscate an obfuscated IC. This research topic is vastly under-explored because of its significant challenges: Over the recent years, a large number of deobfuscation methods have been proposed [10]. The behavior of such highly-nonlinear and strongly-coupling systems is prohibitive for conventional simple models to characterize. The inputs of the runtime estimation problem is the ICs with selected gates obfuscated. Conventional graph neural networks are not intuitive to be applied to such type of varying-structured graph across different instances without significant information loss. Our code and data are available at https://bitbucket.org/submission-repo/anonymous/src/master/.

This work addresses all the above challenges and proposes the first generic framework for deobfuscation runtime prediction, based on Conjunctive Normal Form (CNF) graph representation for obfuscated Circuits. The major contributions of this paper are: **(1)** Proposing a new framework, CNF-Net, for deobfuscation runtime estimation based on graph deep learning.;**(2)** Designing an energy-based neural layer to process varying size of graph data.

2 BACKGROUND AND RELATED WORK

Logic Obfuscation often referred to as logic locking [24] is a hardware security solution that facilitates to hide the Intellectual Property (IP) using key-programmable logic gates. Although obfuscation schemes try to minimize the probability of determining the correct key by an attacker, and avoid making pirated and illegal copies, introducing SAT attack shows that these schemes can be broken [19]. In order to perform SAT attack, the attacker is required to have access to the functional IC along with the obfuscated netlist, and different SAT-hard schemes such as [22, 23] are proposed. Furthermore, new obfuscation schemes that focus on non-Boolean behavior of circuits [21], that are not convertible to an SAT circuit is proposed for SAT resilience. Some of such defenses include adding cycles into the design [16]. By adding cycles into the design may cause that the SAT attack gets stuck in the infinite loop, however, advanced SAT-based attacks such as cycSAT [26] can extract the correct key despite employing such defenses. Before the proposed defense ensures robustness against SAT attacks, the defenders need to run the rigorous simulations which could range from few minutes up to days or even years.

3 THE PROPOSED MODEL: CNF-NET

The ICs deobfuscation can be considered as solving the Boolean satisfiability (SAT) of a conjunctive normal form (CNF) since SAT attack are based on CNF format. Specifically, Fig. 2 exemplifies for us this process as three steps. **Step 1** shows the obfuscated IC where several gates (the two blue gates) have been encrypted into the new blue components with additional key inputs (i.e., $x1$, $x2$, and $x3$) shown in **Step 2**. These new components can be equivalent to the original IC only when the key inputs are correctly inferred. This will be achieved only when the inferred circuit in Step 2 and the original one in Step 1 will always produce the same output value of Y given the same values of the inputs A, B, C, D. Such a problem is routinely formulated as a CNF expression shown in **Step 3**, and the solving of it is a standard SAT problem, which has been proved to be NP-complete [2, 9].

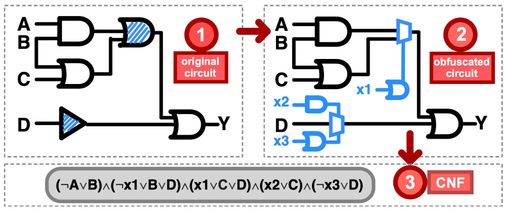


Figure 2: Illustration of transformation from original circuit to obfuscated circuit, and then to CNF representation.

Therefore, the runtime prediction is dependent on the CNF expression. As a standard formula of a Boolean expression, CNF is a conjunction of one or more clauses, where a clause is a disjunction of literals. In other words, different clauses are connected by "AND" operators while each clause consists of one or more literals (or their negations) connected by "OR" operators [18]. Hence, we formally present the problem formulation of this paper as follows: Given the CNF (denoted as CNF_i) of the ith obfuscated IC, the goal of this work is to predict the runtime T_i by a runtime prediction function $f : CNF_i \longrightarrow T_i$, where T_i is typically a non-deterministic value.

As shown in Fig. 3, an obfuscated IC is encoded into the corresponding CNF, which will be mathematically formulated as a bipartite graph, called *CNF bipartite graph*. Following the proposed

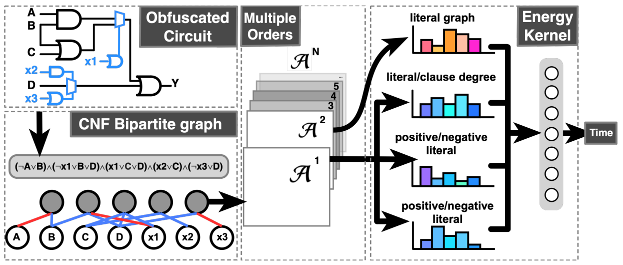


Figure 3: Workflow of CNF-Net: 1) extract CNF graph from obfuscated circuit; 2) derive multiple order information from graph representation; 3) apply energy kernel to aggregate intermediate features;

method, multiple order information is extracted from the CNF bipartite graph by calculating the power series of its adjacency matrix. By leveraging the property of the bipartite graph, the computational cost is significantly reduced. Without loss of generality, the first two orders are considered in this model, and it is easy to extend to any order. After extract raw features of *CNF bipartite graph*, an energy-based[15] kernel is proposed to model the dynamic-size data. This new kernel calculates the energy which identifies the complexity of corresponding CNF bipartite graph.

3.1 CNF Bipartite Graph Representation

The CNF of an obfuscated circuit is modeled as an undirected and signed bipartite graph which uses one node type for clauses and the other for literals. This *CNF bipartite graph* is exemplified in Fig. 4 and defined as $\mathcal{G}(E, V^{literal}, V^{clause})$, where $V^{literal}, V^{clause}$ indicate the set of literal and clause nodes, respectively. The sign of an edge between a literal l and a clause c denotes whether l is negated or not in c. That means, the edge value is: 1 (positively connected), if l is in c, and l is positive; -1 (negatively connected), if l is in c, and l is negative; 0 (disconnected), if l is not in c.

Based on the above formulation, we denote the ith CNF bipartite graph with adjacency matrix $\mathcal{A}_i \in \mathbb{R}^{N_i \times N_i}$, where $N_i = |V_i^l| + |V_i^c|$ is the total number of literal and clause nodes. Typically, the CNFs for different obfuscated of the same circuit (or different circuits) are different, leading to $\mathcal{A}_i \neq \mathcal{A}_j$ givein $i \neq j$. The CNF bipartite graph provides a comprehensive representation of the CNF without information loss, and hence enables the end-to-end frameworks to sufficiently learn any important features automatically. Moreover, we find that the CNF bipartite graph is a powerful representation such that its multi-order version can also capture additional meanings of a CNF. Specifically, here, the 1st- and 2nd-order information is studied first and then expand to higher orders. This multi-order information is used as an immediate input representation of an obfuscated IC to explore the patterns associated with the deobfuscation runtime estimation.

1st and 2nd order CNF bipartite graph information: The 1st order information is the connectivity between literals and clause, i.e., adjacency matrix $\mathcal{A}$. Note that $\mathcal{A}$ is shown on the left side of Fig. 5, which only has zero values for the diagonal blocks and non-zero values in other parts, each of which is an incident matrix. Since the incidence matrix is symmetric in $\mathcal{A}$, only one blue block is sufficient to represent $\mathcal{A}$ without any information loss, leading

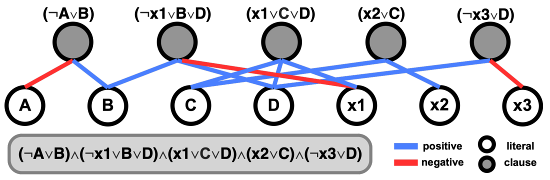
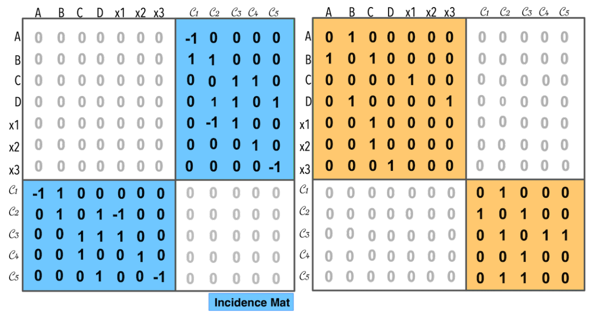


Figure 4: From CNF to 1st order graph.

to significant computational savings. The 2nd order graph holds 2nd order connectivity which can be obtained by taking a square of the adjacency matrix, i.e., $\mathcal{A}^2$. Then the numbers in $\mathcal{A}^2$ indicate the connections only between the same type of nodes. As shown on the right side of Fig. 5, the numbers in the diagonal blocks are all zero, and non-zero otherwise, which exactly the reverse case of the 1st order adjacency matrix. This implies that there is no need to feed the whole $\mathcal{A}^2$ into the model as well. Section 3.2 will show a method which fully utilizes this property to dramatically reduce the computation cost. Following this, the 3rd order adjacency matrix indicated the connectivity with 3rd order neighbors and had zero values again in the diagonal blocks while zeros elsewhere, the same as the 1st order one shown on the left of Fig. 5. Similarly, the 4th order one is the same way as that of the 2nd order one, so on so forth. This means we can save significant amount of computations when considering higher-order graph information in the same way as we discuss the case for 1st- and 2nd-order graph. There is a trade-off between efficiency and order number since more orders mean more information but take more computational resource. Therefore, we only consider the 1st and 2nd order to improve efficiency, and it can be easily extended to any order as needed.

3.2 Energy-based Operators for CNF Graph

Unlike the conventional graphs, the correlation among the neighboring nodes in the CNF bipartite graph does not indicate "proximity" or "similarity" but instead indicates logical relation with signed edges in a variable-size bipartite graph. A novel graph encoder layers have been proposed by leveraging and extending the energy of restricted Boltzmann machine (RBM) [17].



	A	B	C	D	x1	x2	x3	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$
A	0	0	0	0	0	0	0	-1	0	0	0	0
B	0	0	0	0	0	0	0	1	1	0	0	0
C	0	0	0	0	0	0	0	0	0	1	1	0
D	0	0	0	0	0	0	0	0	1	1	0	1
x1	0	0	0	0	0	0	0	0	-1	1	0	0
x2	0	0	0	0	0	0	0	0	0	0	1	0
x3	0	0	0	0	0	0	0	0	0	0	0	-1
$\mathcal{C}_1$	-1	1	0	0	0	0	0	0	0	0	0	0
$\mathcal{C}_2$	0	1	0	1	-1	0	0	0	0	0	0	0
$\mathcal{C}_3$	0	0	1	1	1	0	0	0	0	0	0	0
$\mathcal{C}_4$	0	0	1	0	0	1	0	0	0	0	0	0
$\mathcal{C}_5$	0	0	0	1	0	0	-1	0	0	0	0	0

Incidence Mat

	A	B	C	D	x1	x2	x3	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	0	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0	0	0
D	0	1	0	0	0	0	1	0	0	0	0	0
x1	0	0	1	0	0	0	0	0	0	0	0	0
x2	0	0	1	0	0	0	0	0	0	0	0	0
x3	0	0	0	1	0	0	0	0	0	0	0	0
$\mathcal{C}_1$	0	0	0	0	0	0	0	0	1	0	0	0
$\mathcal{C}_2$	0	0	0	0	0	0	0	1	0	1	0	0
$\mathcal{C}_3$	0	0	0	0	0	0	0	0	1	0	1	1
$\mathcal{C}_4$	0	0	0	0	0	0	0	0	0	1	0	0
$\mathcal{C}_5$	0	0	0	0	0	0	0	0	1	1	0	0

Figure 5: A simple example showing (left): adjacency matrix of 1st order graph: $\mathcal{A}$, which models the example in Fig. 4; and (right): adjacency matrix of 2nd order graph: $\mathcal{A}^2$. There are 5 clauses: clause $C_1 = \{\neg A, B\}$, clause $C_2 = \{\neg x1, B, D\}$, clause $C_3 = \{x1, C, D\}$, clause $C_4 = \{x2, C\}$, clause $C_5 = \{\neg x3, D\}$

RBMs for CNF bipartite graphs: Inspired by RMB, $\boldsymbol{v}$ and $\boldsymbol{h}$ are the representations of a literal and a clause, respectively. Similarly, an energy form is defined for characterizing a CNF: $E = -\alpha \cdot E_{literal} - \beta \cdot E_{clause} - \gamma \cdot E_{inter}$, where $E_{literal}$, E_{clause} and E_{inter} are the energies of literals, clauses, and their connections, and α, β, γ are the weights of them, respectively. Since SAT runtime estimation over CNF is a highly nonlinear process, traditional linear function has been generalized by us into a new non-linear version:

$$E = f_\Phi(E_{literal}, E_{clause}, E_{inter}), \tag{1}$$

where f_Φ is a neural network function controlled by parameter Φ. In the following, we study bipartite connection energy E_{inter} first and then $E_{literal}, E_{clause}$ in turn. Based on RBM, E_{inter} is defined as linear function of literals: $E_{inter} = \sum_m \sum_n v_m w_{m,n} h_n$, where v_m is a literal and h_n is a clause in one single CNF bipartite graph $\mathcal{G}_i$. However, E_{inter} is not necessarily a sum function. Therefore, we generalize E_{inter} inspired by generalizing convolutional graph layers into bipartite graphs. However, most existing graph deep learning operators focus on graphs with fixed topology while the size and topologies of CNF bipartite graph vary across different instances dramatically. To solve this problem, we design a kernel for aggregating interaction information in one graph. Specifically, a d-dimensional vector of pseudo-coordinates is associated with $[\boldsymbol{v}, \boldsymbol{h}]$, we define a weighting kernel $Z_\Theta(\cdot, \cdot)$, so that for one CNF bipartite graph $\mathcal{G}_i$, we have:

$$E_{inter} = \sum_m \sum_n Z_\Theta(\mathcal{E}(\boldsymbol{v}_m, \boldsymbol{h}_n)) \cdot \mathcal{E}(\boldsymbol{v}_m, \boldsymbol{h}_n), \tag{2}$$

where $Z_\Theta(\cdot)$ projects the $[\boldsymbol{v}, \boldsymbol{h}]$ into a new value as the weight of $[\boldsymbol{v}, \boldsymbol{h}]$, and $\mathcal{E}(\boldsymbol{v}_m, \boldsymbol{h}_n)$ represents the interaction or edge value between $\boldsymbol{v}_m$ and $\boldsymbol{h}_n$ such as 1, -1 or 0. Note that Z_Θ function is implemented by neural networks and controlled by fixed-size parameters Θ. Similarly, we further generalize $E_{literal}, E_{clause}$ as:

$$E_{literal} = \sum_m Z_\Theta(\boldsymbol{v}_m) \cdot \boldsymbol{v}_m \text{ and } E_{clause} = \sum_n Z_\Theta(\boldsymbol{h}_n) \cdot \boldsymbol{h}_n, \tag{3}$$

where $\boldsymbol{v}$ and $\boldsymbol{h}$ indicate attributes of literal and clause respectively. Therefore, the final model for CNF is:

$$E = f_\Phi(\sum_m Z_\Theta(\boldsymbol{v}_m) \cdot \boldsymbol{v}_m, \sum_n Z_\Theta(\boldsymbol{h}_n) \cdot \boldsymbol{h}_n, \sum_m \sum_n Z_\Theta(\mathcal{E}(\boldsymbol{v}_m, \boldsymbol{h}_n)) \cdot \mathcal{E}(\boldsymbol{v}_m, \boldsymbol{h}_n)), \tag{4}$$

Energy model for 1st-order graph operators: Eq. (4) above dose not consider the sign of the edges between literals and clauses. Hence, positive and negative information is encoded separated: $E = \{E^+, E^-\}$. To capture the sign information, the corresponding incidence matrix $INC \in \mathbb{R}^{|V^{literal}| \times |V^{clause}|}$ is utilized:

Normalized positive and negative edge distribution in clause scope (NPNC): we count positive and negative edges for each clause, and take normalization of both positive and negative counts, so there are two values for each clause, i.e., c^{pos} and c^{neg}. If there exist $|V^{clause}|$ clauses, there will be $2|V^{clause}|$ number of features:

$$\left\langle c^+_{clause}(0), c^-_{clause}(0) \right\rangle, \left\langle c^+_{clause}(1), c^-_{clause}(1) \right\rangle, \ldots, \tag{5}$$

which can be obtained by column-wise summation on positive values only and negative only of incidence matrix.

Normalized positive and negative edge distribution in literal scope (NPNL): Similarly, positive and negative degrees are counted for each literal, and the normalization per literal is taken. there will be $2|V^{literal}|$ number of features. In sum, positive- and negative-valued edges are treated as separate operators, so we have 2 feature maps for NPNC after normalization.

	MSE						Pearson						Spearman					
	c432	c499	c880	c1355	c1908	c2670	c432	c499	c880	c1355	c1908	c2670	c432	c499	c880	c1355	c1908	c2670
EN	17.422	16.836	18.488	19.873	17.550	17.223	0.333	0.112	0.436	0.046	0.301	0.440	0.625	0.491	0.726	0.609	0.553	0.704
LARS	17.454	16.421	18.546	19.872	17.593	17.313	0.314	0.038	0.331	-0.000	0.393	0.332	-0.123	0.215	**0.888**	0.150	0.437	**0.895**
LASSO	17.342	17.120	18.358	19.896	17.407	17.124	0.572	0.076	0.515	-0.038	0.441	0.533	0.417	0.353	0.377	0.300	0.374	0.622
LR	17.315	17.518	18.281	19.861	17.591	16.744	0.582	-0.088	0.550	0.105	0.355	0.727	0.426	0.162	0.215	0.208	0.347	0.122
OMP	17.421	16.976	18.521	19.881	17.433	17.304	0.314	0.038	0.331	-0.000	0.393	0.332	-0.123	0.215	**0.888**	0.150	0.437	**0.895**
PAR	17.452	16.226	18.618	19.876	17.596	17.409	0.150	0.172	0.306	0.105	0.220	0.318	**0.794**	**0.927**	**0.869**	**0.947**	**0.812**	**0.871**
Ridge	17.379	17.032	18.467	19.879	17.469	17.154	0.489	0.079	0.445	0.025	0.384	0.503	0.526	0.394	0.462	0.477	0.371	0.594
SGD	68.412	72.478	65.875	73.447	72.771	75.613	-0.159	0.145	0.226	-0.121	0.202	-0.332	-0.808	**0.887**	**0.812**	-0.941	**0.894**	-0.895
SVR	17.446	16.259	18.613	19.876	17.612	17.450	0.737	0.092	0.375	0.016	0.080	0.213	0.411	0.533	0.476	0.540	0.775	**0.839**
Theil	20.350	20.089	18.512	19.941	20.063	17.106	0.012	-0.038	0.546	0.033	0.007	0.615	0.354	0.365	0.288	0.562	0.551	0.240
DistNet	174.0315	18.406	9.3729	12.8988	22.8172	7.0459	0.175	0.638,	0.5472	0.265	**0.8915**	0.4034	-0.1229	**0.8043**	0.774	**0.8903**	0.5021	0.0995
DCNN	**4.458**	**3.897**	**7.431**	**5.356**	**4.353**	**6.312**	-0.061	0.034	0.203	-0.099	-0.021	-0.031	0.030	-0.005	0.213	-0.037	-0.043	-0.030
CNF-Net	5.758	4.164	7.719	6.602	20.063	7.273	**0.736**	**0.686**	**0.770**	**0.878**	**0.710**	**0.855**	**0.794**	**0.858**	**0.888**	**0.944**	0.5519	**0.884**

Table 1: Prediction performance on 6 benchmark circuits: MSE, Pearson and Spearman correlation

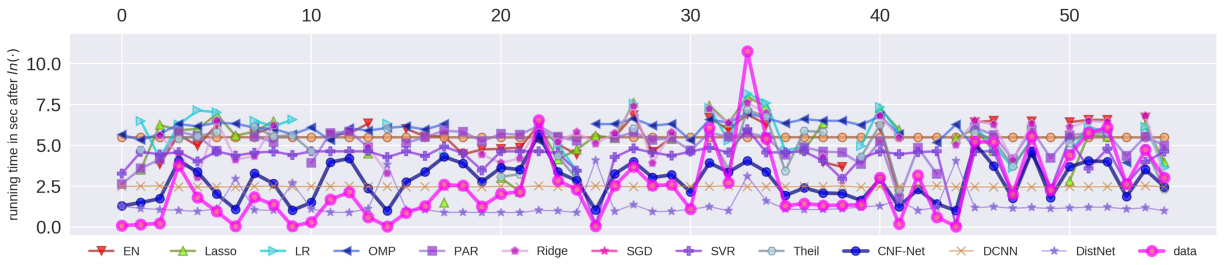


Figure 6: Prediction performance on samples from two datasets c432 and c880(negative values are removed): x-axis is the data index and y-axis denote predicted runtime compared with real runtime(label of*data*)

Energy Model for the 2nd-order graph operators The 2nd order of graph with adjacency matrix $\mathcal{A}^2$ denotes the literal-wise and clause-wise mutual information, which corresponds to the top left and bottom right block respectively in the right subfigure of Fig. 5. To further emphasize this important trait and reduce the computational complexity, our model only distinguish zero with non-zero values in 2nd order graph information, which means that we only consider if: two literals co-appear in the same clause or two clauses share the same literal.

4 EVALUATION

For the experimental setup, we have used the ISCAS-89 benchmarks (http://www.pld.ttu.ee/~maksim/benchmarks/iscas85/verilog/), and obfuscation instances on 6 different circuits are selected as 6 benchmark datasets. The synthesized netlist is further converted to the bench format as per the SAT-solver requirements [18]. The in-house developed script replaces the gates with the Look-up-table of size 2, as described in [8]. The output of the script is the obfuscated bench file along with the adjacency matrix. The time taken by the SAT-solver to find the correct key is the deobfuscation time. We compare against several state of art regression models (https://scikit-learn.org/stable/modules/linear$_m$odel.html): Linear Regression(LR),Passive Aggressive Regression(PAR) [3], LASSO [20], Epsilon-Support Vector Regression (SVR), Ridge Regression (RR) [14], Elastic Net(EN) [27], Orthogonal Matching Pursuit (OMP) [12], SGD Regression, Least Angle Regression (LARS) [5], Theil-Sen Estimators(Theil) [4]. In addition, a neural network for predicting runtime work DistNet [6] is employed.

Performance analysis Table (1) shows the MSE after taking ln(·), and two correlation for all the methods on 6 datasets. DCNN achieved the best MSE score throughout all datasets, while our method is the second best. MSE scores of regression models are much higher, which shows their ineffectiveness. On the contrary, this observation implies the effectiveness of the proposed methods in considering and utilizing the graph features. Regarding correlation, CNF-Net outperformed all the other baselines. This advantage confirms our model can well identify the trend of runtime. In terms of the correlation between predicted runtime and real runtime, DCN's scores are significantly small, i.e., less than 20% of the Pearson coefficient. While CNF-Net achieves around 70% correlation and up to 87% of Pearson coefficient at c1355 dataset. The same advantage of CNF-Net shows up in Spearman evaluation: the correlation score is more than 80% for five datasets, which verifies its effectiveness in estimating the runtime trend. The performance on Spearman of PAR is similar to CNF-Net, which also achieved high correlations. Overall, CNF-net is balanced and has good performance in predicting runtime task. However, the inconsistent performance of DCNN using different metrics shows the insufficiency of these metrics. Therefore, random samples were evaluated on all the methods to illustrate the differences among them. As shown in Fig. (6), 56 samples from dataset c432 were used to perform evaluation one by one. DCNN predicted runtime by a constant value which is close to the mean of the distribution. This can help DCNN to achieve low MSE, but it is useless in practice since DCNN cannot distinguish high values from low values. CNF-Net can almost match the trend of these samples except extremely high values.

5 CONCLUSION

This paper presents a novel graph neural networks framework to identify the security level of obfuscated ICs by predicting runtime of SAT attack. Our work proposed to employ a bipartite graph as a representation of obfuscated ICs and characterize this graph by multi-order information without any information loss. An energy-based kernel is designed to aggregate dynamic features of the graph representation. By utilizing the special properties of the bipartite graph under our case, the computational cost is further reduced. Extensive experiments on several benchmarks demonstrated the advantageous performance of the proposed model over the existing method for runtime prediction task.

REFERENCES

[1] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1993–2001.

[2] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 151–158.

[3] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *Journal of Machine Learning Research* 7, Mar (2006), 551–585.

[4] Xin Dang, Hanxiang Peng, Xueqin Wang, and Heping Zhang. 2008. Theil-Sen estimators in a multiple linear regression model. *Olemiss. edu* (2008).

[5] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.

[6] Katharina Eggensperger, Marius Lindauer, and Frank Hutter. 2018. Neural Networks for Predicting Algorithm Runtime Distributions.. In *IJCAI*. 1442–1448.

[7] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. 2015. Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes.. In *NDSS*.

[8] Hadi Mardani Kamali, Kimia Zamiri Azar, Kris Gaj, Houman Homayoun, and Avesta Sasan. 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)* (2018), 1–6.

[9] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.

[10] Soroush Khaleghi and Wenjing Rao. 2018. Hardware Obfuscation Using Strong PUFs. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 321–326.

[11] Duo Liu, Cunxi Yu, Xiangyu Zhang, and Daniel Holcomb. 2016. Oracle-guided incremental SAT solving to reverse engineer camouflaged logic circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*. IEEE, 433–438.

[12] Stéphane G Mallat and Zhifeng Zhang. 1993. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing* 41, 12 (1993), 3397–3415.

[13] IHS Technology Press Release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry. [n. d.]. https://technology.ihs.com/405654/top5-most-counterfeited-parts-represent-a-169-billion-potentialchallenge-for-global-semiconductor-market, 2. ([n. d.]). http://www.test.org/doe/

[14] Andrew Y Ng. 2004. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 78.

[15] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. 2007. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*. 1137–1144.

[16] Shervin Roshanisefat, Hadi Mardani Kamali, and Avesta Sasan. 2018. SRCLock: SAT-Resistant Cyclic Logic Locking for Protecting the Hardware. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI '18)*.

[17] Paul Smolensky. 1986. *Information processing in dynamical systems: Foundations of harmony theory*. Technical Report. Colorado Univ at Boulder Dept of Computer Science.

[18] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 137–143.

[19] Pramod Subramanyan, Sayak Ray, and Sharad Malik. 2015. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 137–143.

[20] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[21] Y. Xie and A. Srivastava. 2017. Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.

[22] Y. Xie and A. Srivastava. 2018. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018), 1–1.

[23] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*.

[24] M. Yasin, J. J. Rajendran, O. Sinanoglu, and R. Karri. 2016. On Improving the Security of Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 9 (Sept 2016), 1411–1424.

[25] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan JV Rajendran, and Ozgur Sinanoglu. 2017. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1601–1618.

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

[27] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*

67, 2 (2005), 301–320.

APPENDIX A ALGORITHM

Algorithm 1: CNF-Net

```
Input: $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \dots \mathcal{G}_i \dots, \mathcal{G}_N\}$, the real runtime $T_i$ for
       instance $\mathcal{G}_i$
Output: a neural network function with parameters $\Phi$, $\Theta$, parameters
        of fully connected layers($\tau$), and distribution parameter $\sigma$
1  // data preparing
2  transform from obfuscated circuit $\mathcal{G}_i$ into CNF representation $CNF_i$
     and derive bipartite graph
3  extract adjacency matrix $\mathcal{A}$ from this bipartite graph and calculate
     power series : $\{\mathcal{A}^1, \mathcal{A}^2, \dots, \mathcal{A}^N\}$        ▷ Section (3.1)
4  compute distribution features based on power series                 ▷ Eq. (3)
5  $\theta = \{\Theta, \Phi, \tau, \mu, \sigma\}$
6  initialize $\theta$ with standard Gaussian
7  // update CNF-Net
8  repeat
9      apply the energy kernel on laterals and clauses.                 ▷ Eq. (3)/
10     apply the energy kernel on the connection between laterals and
         clauses                                                          ▷ Eq. (2)
11     calculate the overall energy E and then feed E into fully
         connected layer                                                  ▷ Eq. (1)
12     get a intermediate predicted value $t$, and apply distribution $e^t$ as
         predicted time calculate residues $\delta = T - e^t$
13     compute derivatives to update parameters: $\theta \longleftarrow \theta + \alpha \nabla_\theta \delta$,
         where $\alpha$ is learning rate
14 until $\delta$ convergence;
```

The algorithm (1) first prepare CNF graph adjacency as obfuscated circuit representation (line 2). Then power series are extracted based on CNF graph (line 3). The model computes feature maps based on power series 4. All model parameters are initialized by Gaussian distribution (line 5 to 6). In each training iteration, energy-based kernel is applied to literals, clauses and their interaction (line 9 to 10). The aggregation of energy is treated as features of the targeted obfuscated IC. To model the variance of runtime for similar or exactly the same instance, a distribution kernel is applied in the last layer. This distribution kernel can be replaced with any other suitable distributions such as logarithmic or exponential kernel. The algorithm (1) presents such idea using normal distribution. Then typical parameter optimization(i.e., Adam) of deep neural networks is employed.

APPENDIX B RELATIONSHIP WITH SPATIAL GRAPH NN

As a state-of-art of the spatial graph neural networks model, the superiority of DCNN [1] beyond the other graph deep learning models is the capacity in handling graphs with variable sizes and considering multiple orders, which is also one advantage of our model. Our study demonstrates that DCNN is a special case of our CNF-NET:

Lemma B.1. *CNF-Net is a generalization of DCNN, while DCNN is a special case of CNF-Net when setting the feature aggregation to mean function.*

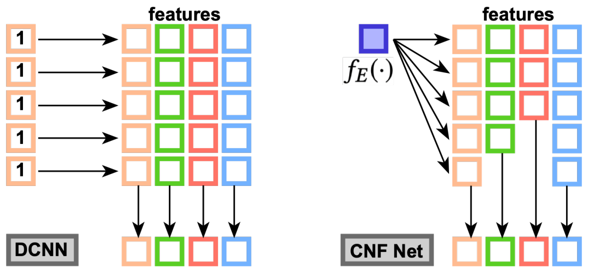


Figure 7: Comparison with DCNN

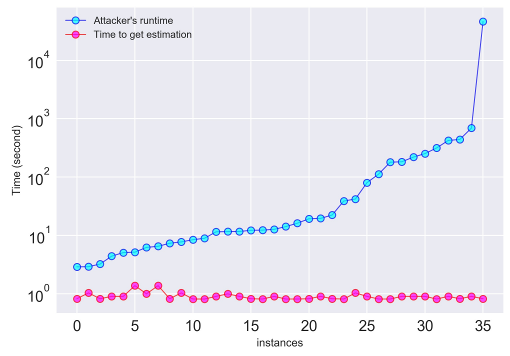


Figure 8: Real runtime compared with prediction time

Proof. DCNN can be extended to whole graph modeling by taking the mean activation over the features on diffusion $P_t^* X_t$:

$$Z = f(W^c \odot 1_{N_t}^T P_t^* X_t / N_t) = f(W^c \odot \underbrace{(\frac{1}{N_t})_{N_t}^\top}_{\text{aggregation}} P_t^* X_t), \tag{6}$$

where $(\frac{1}{N_t})_{N_t}$ is a $N_t \times 1$ vector of value $\frac{1}{N_t}$), t indicates the graph index, and P^* is power series of adjacency matrix. Following the same representation, we can rewrite CNF-Net as:

$$Z = f(W^c \odot \underbrace{f_E(\cdot)}_{\text{aggregation}} P_t^* X_t), \tag{7}$$

where $f_E(\cdot)$ represents a vector of $\left[Z_\Theta(\phi(i))\right]_{i=0}^{d_{feat}-1}$, and d_{feat} indicate dimension of a feature and $\phi(i)$ is the i-th value along a feature. Therefore $f_E(\cdot)$ is a vector of dynamic size. □

APPENDIX C PREDICTION EFFICIENCY

Our task is to predict runtime efficiently. Otherwise, there is no practical value for this method. As shown in Fig. (8), prediction time (pink) using the proposed model is compared with real runtime in blue (i.e., running SAT-based attack). CNF-Net took a stable and little time to predict the runtime, which demonstrates its efficiency.