# Analysis of Security of Split Manufacturing Using Machine Learning

Wei Zeng, Boyu Zhang, and Azadeh Davoodi, *Senior Member, IEEE*

***Abstract*—This work is the first to analyze the security of split manufacturing using machine learning, based on data collected from layouts provided by industry, with 8 routing metal layers and significant variation in wire size and routing congestion across the layers. Many types of layout features are considered in our machine learning model, including those obtained from placement, routing, and cell sizes.**

**Since the runtime cost of our basic machine learning procedure becomes prohibitively large for lower layers, we propose novel techniques to make it scalable with little sacrifice in the effectiveness of the attack. Moreover, we further improve the performance in the top routing layer by making use of higher-quality training samples and by exploiting the routing convention. We also proposed a validation-based proximity attack procedure, which generally outperforms our recent prior work.**

**In the experiments, we analyze the ranking of the features used in our machine learning model and show how features vary in importance when moving to the lower layers. We provide comprehensive evaluation and comparison of our model with different configurations, and demonstrate dramatically better performance of attacks compared to the prior work.**

## I. Introduction

Chip fabrication has now spread across the globe, with over 90% of the world's foundry capacity controlled by non-US companies. In some cases, use of off-shore fabrication facilities may be inevitable due to the need for high volume production and access to advanced facilities. Moreover, in the era of Internet of Things (IoT), there are billions of connected devices which may be fabricated off-shore and vulnerable to hacking. In the light of such IoT attacks, it is crucial to ensure these everyday devices are securely fabricated.

Split manufacturing is an IC fabrication model which enables secure use of a high-end but untrusted foundry. In this model, only partial information about the chip is sent to the untrusted foundry and the chip is only partially fabricated [2], [3], [9], [13]. For a layout given as a network of connected transistors, the partial information corresponds to the complex steps of fabricating the transistor layer and a subset of metal layers that are immediately above it (i.e., Front End of Line, FEOL). Fabrication of the rest of the connections, captured by the higher metal layers (i.e., Back End of Line, BEOL), won't require a complex fabrication process and can be done by a smaller, trusted company.

The majority of recent studies have suggested that split manufacturing is inherently unsecure while not making reasonable assumptions about complexities and sizes of designs and physical synthesis in modern fabrication processes. Therefore the focus of many recent works has been on obfuscation, for example scrambling the locations of broken nets at the split layer to make proximity and other attacks more challenging [3], [7], [9], [13], [16]. Recent work [5], which considered these modern layout factors, showed a popular proximity attack [9] is not effective. However, the technique in [5] had limitations such as being constrained to place and route layout features, and the use of simple linear regression for modeling while simultaneously considering all designs (without separation of testing and training cases).

In this work, we use the identical setup as [5] but include more layout features. We also use machine learning for modeling while ensuring separation of training and testing data sets. Similar to [5], the emphasis is to find a small list of candidates for each broken net with a high accuracy to include the actual match. Our contributions are listed below.

- We incorporate various layout features including placement, routing, cell sizes, and cell pin types for machine learning. We propose novel techniques for effective realization of training and testing in machine learning.
- Since the runtime of our basic machine learning procedure becomes prohibitively large for lower layers, we propose novel ways to make it scalable, achieving significant runtime improvement without much sacrifice in the effectiveness of the attack.
- We improve the performance of attack for the top routing layer by using high-quality training samples and exploiting the routing conventions.
- We conduct a comprehensive study on proximity attack. Based on the study, we propose a validation-based procedure to improve the success rate of proximity attack.
- We study the ranking of features with each feature associated with a weight signifying its importance. Using the ranked features, we study relative importance of the features across the layers. We show routing features are the most important ones, and as we move to lower layers, more features become important.
- Based on the feature ranking, we show how design obfuscation can make the attack more difficult by applying artificial random noises to two of the most important features.

All experiments are conducted based on data collected from design layouts released by industry [12], featuring 9 routing metal layers with significant variation in wire size ($4\times$) and significant variation in routing congestion across the layers.

In our experiments we show that, for example, for split

This research was supported by Award Number 1812600 from National Science Foundation and by Semiconductor Research Corporation.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin–Madison, Madison, WI 53706 USA (e-mail: wei.zeng@wisc.edu; bzhang93@wisc.edu; adavoodi@wisc.edu).

layer 8, the size of list of candidates for the broken nets is on-average 2–3% of that in [5] while maintaining the same likelihood of including the actual match in the list; with the same size of list of candidates, the likelihood of including the actual match in the list is 99.99% compared to 42.72%. The improvements to the scalability of our machine learning procedure reduced the average attack runtime on layer 4 from 5.31 hours to around 1 hour. Overall, this is the first work to apply machine learning to study security of split manufacturing and show significant improvement in the effectiveness of the attack.

## II. Attack Model and Prior Work

### A. Attack Model and Preliminaries

In our model of split manufacturing, an untrusted foundry has received a layout file which contains information about the transistor layer and some metal layers that are immediately above it (FEOL). The goal of the untrusted party is to guess the missing connections in the higher layers (BEOL) using the FEOL information. The layout file, which is typically described in the GDSII format, allows quick generation of a gate-level description of the partially-connected network. It includes information on where each cell is placed, as well as the characteristics of each cell which includes factors such as cell type, electrical properties, drive strength, and pin shape. The route fragments of the partial network are also completely specified on each layer corresponding to the FEOL. In this attack model, the goal of the untrusted party is to guess the match for each incomplete connection.

A **split layer** is a via layer at which the split is made. For example, split layer 6 means the split at via layer 6, separating metal layers 6 and 7. In this case, the untrusted foundry (i.e. attacker) gets the layout of placed cells and macros, all wires up to metal layer 6, and all vias up to via layer 6. Layout on and above metal layer 7 is not available to this foundry.

We designate the point where a net is broken on the split layer as a **virtual pin**, which we refer to as *v-pin* throughout the paper. In the context of VLSI, a v-pin is essentially a via at the split layer. This is in contrast to the input/output pins of the standard cells which are typically located on layer M1. Each v-pin may connect through a route fragment to one or more pins at the placement layer.

### B. Prior Work

Many recent works on split manufacturing ignore important factors which directly impact the difficulty of the attack. For example the work [11] uses very small designs, with at most 6 500 gates. Ten metal layers are assumed for 45 nm technology. It is not clear why so many metal layers were used for such small designs. While the authors use Cadence tools for synthesis, it is unclear how the parameters of the tool were controlled to generate a fair layout; for example, they mention "use of *all* metal layers" for all benchmarks for fair comparison from a split manufacturing perspective. If this is influenced by the user, then it does not fairly capture the behavior of typical routing algorithms. In general a minimum number of metal layers are picked by the designer, and more congestion is seen in the lower layers so picking many layers and evenly distributing the routes across them is not fair and may significantly simplify the split manufacturing attack. Wire sizes and their variations across the layers are not mentioned in [11], which is a very important factor in how congestion gets spread across the layers. (Higher congestion in the split layer makes it more likely for the two segments of a broken net to be further apart from each other on that layer and makes the attack more challenging.) In [9], it is assumed that only I/O pins are cut by the split layer, which is clearly not a realistic assumption. Similar issues exist in many other prior works.

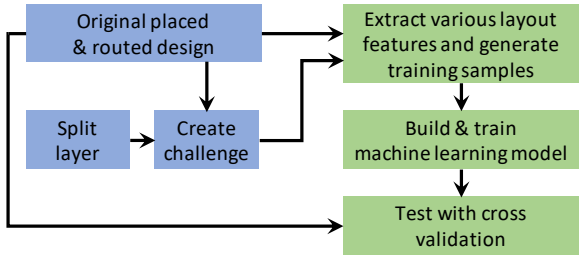



Fig. 1. Flow chart of our modeling process.

Moreover, with smaller designs, the number of cut nets at the split layer will be significantly small. This allows use of computationally intensive algorithms which can be infeasible in practice. For example, an interesting network flow model in [13] and pattern-matching-based layout recognition attacks in [17] are applied to designs with at most 18 000 gates. The actual number of edges in the graph for an 18K design is much larger (potentially exponential order) because of considering many matching candidates simultaneously. These models are infeasible to apply in our setup with designs of at least a few hundred thousand gates. It requires significant effort to develop scalable variations of these algorithms.

Recently, the work [5] uses large layouts with a setup released by industry to study split manufacturing. While these designs consider factors such as wire size variation, and proper use of the place and route tool (as supervised/released by industry), their study was limited in many aspects. First, it only considered a small number of place and route features. Moreover, its modeling process was based on simple linear regression and across all designs (without separation of testing and training), leaving significant room for improvement as we show in this work.

Since existing techniques are generally independent of our machine learning framework in this paper, attackers could combine them, where possible, for even better performance of attack, which is out of scope of this paper.

## III. Machine Learning Framework

In this paper, we propose a machine learning framework to carry out the attack of split manufacturing. To simplify discussion, in this paper we assume this framework is the only tool available to the attacker, whereas in reality, this would be only a part of effort in reverse engineering, and the attackers may combine this framework with other knowledge and techniques for potentially better results.

Fig. 1 shows a high-level overview of our modeling process. First, a challenge instance is created from a placed and routed

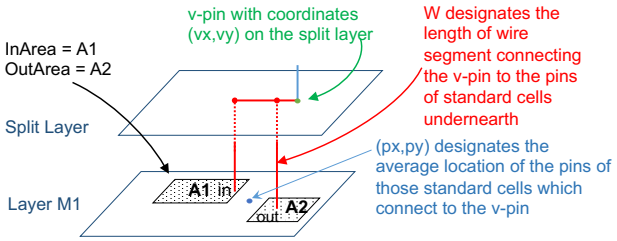



Fig. 2. Layout feature extraction for each v-pin.

layout by cutting it at the split layer and only looking at the FEOL view. Next, a set of layout features are collected for each v-pin, which include features from placement, routing, cell areas, and cell pin types. Using these features, next, machine learning samples are generated which are fed into the training process. Each sample includes information for a *pair* of v-pins which may or may *not* be matching each other. (This step will be explained in detail later.) Sample generation is done on a subset of the designs which are designated for training. Machine learning then builds a classifier using the training samples. Cross validation is used for evaluation which ensures validation of the model is done on the rest of the data which was not used for training.

We first explain how the layout features are collected for individual v-pins, then explain how samples are generated for training, and discuss the details of the modeling process. Finally, we discuss adjustments to the machine learning framework to make it scalable when applying to lower split layers, as well as techniques to improve the results and their interpretability. We also conduct a comprehensive study of performing proximity attack based on the results of machine learning.

### A. Feature Extraction for Individual V-Pins

For each v-pin $v$ (in the designs used for training), we extract the following layout features as shown in Fig. 2. Let $(vx, vy)$ denote the v-pin's coordinates on the split layer. We compute wirelength $W$ for the route fragment that connects $v$ to one or more pins of standard cells on the underneath placement layer. We also calculate the location where the v-pin connects on the placement layer, which we denote by $(px, py)$. If the connection is to multiple pins on the placement layer, the location is computed by averaging the coordinates of pins of the standard cells that connect to $v$.

Among the standard cells that connect to $v$, we sum the areas of those which connect to $v$ through an input pin. We denote this by `InArea`. Similarly, we compute an `OutArea` designating the sum of areas of the standard cells which connect to $v$ through an output pin. Fig. 2 shows an example of how the above quantities are calculated. We note these features together capture characteristics from placement, routing, cell areas, and pin types.

The intuition of defining `InArea` and `OutArea` is to enable accounting for driver strength during machine learning. Driver strength is highly correlated with the cell area. Each cell has a maximum output load that it can drive. The area measurements will be used during machine learning when a pair of v-pins are considered as potential match. Similarly, the reason wirelength $W$ is recorded is to enable identifying cases when a pair of considered v-pins result in an unrealistic combined wirelength that may impact timing, which will be explained in detail in Section III-B.

For each v-pin, we also extract the congestion measurements for placement and routing, denoted by $PC$ and $RC$, respectively. These measurements were introduced in [5] measuring the congestion around the neighborhood of a v-pin. The placement congestion $PC$ is defined as the pin density around the pin that connects to the target v-pin. The routing congestion $RC$ is defined as the v-pin density around the target v-pin. These congestion measurements will also be used to extract features in machine learning, as will be described in Section III-B.

### B. Sample Generation

After the layout features for individual v-pins are extracted, we prepare the samples to feed into the subsequent training step. Specifically, for each v-pin, we first create a pair by recording the index of its (correctly) matching v-pin. This serves as a "positive sample." We then create equal number of samples of non-matching v-pin pairs which we denote as "negative samples." This is done by randomly picking v-pins which are not a match, from the training data set. (We explain later how we divide our benchmarks into testing and training data sets). Note that there are much more non-matching v-pin pairs (negative samples) than matching pairs (positive samples) in a typical design. When dealing with such unbalanced data sets, using equal numbers of positive and negative samples in training as described is shown to be essential for effective modeling [4][1].

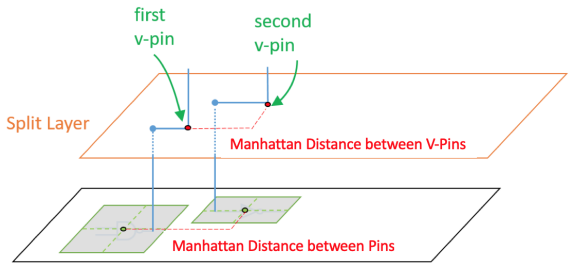
Next, for each pair (matching or non-matching), we record various corresponding layout features. This uses the individual v-pin features which were explained before. Specifically, for a v-pin pair $(v_1, v_2)$ we record the following to generate a sample.

- `DiffPinX` = $|px_1 - px_2|$: This feature records the difference in the $x$-coordinates of the pins on the placement layer which connect to the two v-pins.
- `DiffPinY` = $|py_1 - py_2|$: Same as the previous feature except calculated using the $y$-coordinates.
- `ManhattanPin` = $|px_1 - px_2| + |py_1 - py_2|$: Same as the previous feature except it is calculated based on the pin locations on the placement layer.
  — It is believed that the placer and router tend to minimize the wirelength for less routing congestion and easier timing closure. `ManhattanPin` is the minimal possible wirelength (i.e. the lower bound) connecting the two pins, with `DiffPinX` and `DiffPinY` being the components in $x$ and $y$ directions, respectively. Intuitively, if these features of a certain Vpin pair are too large, this pair is less likely to be a match. This feature is included to allow for consideration of a placement level proximity, which previous work has associated with finding a correct match [5]. Including the Manhattan distance as separate features

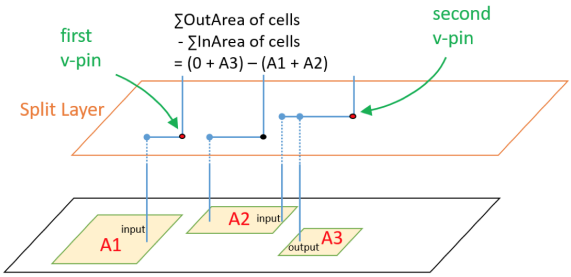
[1]Those v-pin pairs which connect output pins of different cells are illegal and we exclude from both positive and negative samples.

(besides the difference in individual $x$- and $y$-coordinates) gives the machine learning algorithm more flexibility to consider the proximity between the v-pins as a factor in matching the correct v-pin.

- `DiffVpinX` = $|vx_1 - vx_2|$: This feature records the difference in the $x$-coordinates of the two v-pins.
- `DiffVpinY` = $|vy_1 - vy_2|$: Same as the previous feature except calculated using the $y$-coordinates.
- `ManhattanVpin` = $|vx_1 - vx_2| + |vy_1 - vy_2|$: This feature records the Manhattan distance between two v-pins.
  — These features share a similar rationale to the previous three features, except that we consider the coordinates of v-pins instead of pins. It is observed that a candidate v-pin in close proximity is more likely to be a match [5]. Since the attacker already knows the layout below the split layer, `ManhattanVpin` (plus the known wirelength below the split layer) gives a better lower bound of wirelength connecting the Vpin pairs. Besides, when the split layer separates only the top metal layer (split layer $= 8$ in this paper), we are more likely to have a zero difference in either $x$ and $y$ direction, as the top metal layer (the only unknown layer to the attacker) is usually either vertically or horizontally routed. This fact further eases the classification in that particular split layer.
- `TotalWirelength` = $W_1 + W_2$: This is the known wirelength connecting the v-pin pair below the split layer.
  — Since the wirelength of each net impacts timing, it is important to ensure the combined wirelength of the considered v-pin pair is not prohibitively large (which will be decided by the machine learning algorithm based on typical characteristics seen in the training data). In other words, matching pairs usually have smaller values in this feature than non-matching ones.
- `TotalArea` = $InArea_1 + InArea_2 + OutArea_1 + OutArea_2$: This records the sum of cell areas connecting to the two v-pins.
- `DiffArea` = $(OutArea_1 + OutArea_2) - (InArea_1 + InArea_2)$: This feature calculates the area difference of the driver cells from its loads. Note, one of $OutArea_1$ or $OutArea_2$ must be zero. Otherwise the pair is illegal and is disregarded.
  — `TotalArea` and `DiffArea` together help determine if a driver cell has sufficient strength for the load introduced by the v-pin pair. See Section III-A. Fig. 3(a) and (b) illustrate some of the above features.
- `PlacementCongestion` = $PC_1 + PC_2$: Congestion measurement around the pin at placement layer, which measures the density of pins around the pin that is connected to the target v-pin.
- `RoutingCongestion` = $RC_1 + RC_2$: Congestion measurement around the v-pin at the split layer, which measures the density of v-pins around the target v-pin within certain neighborhood.
  — These two congestion measurements are based on the intuition that, if the congestion around the two v-pins is too high, they are not likely to be a match, and in highly congested regions, matching v-pins are more likely to be farther away from each other.




(a) Manhattan distance between v-pins and between pins of standard cells.




(b) Standard cell area based features.

Fig. 3. Illustration of some features for pairs of v-pins.

We will plot the data distributions of these extracted features in Section IV-A, so that interested readers can gain a better understanding about them. We believe these features are not specific to any circuit, because the intuitions behind them all come from general ideas in physical design.

Overall, each sample in the training and testing data sets represents a v-pin pair and includes the above 11 layout features, with a binary target value indicating if the pair is a match or not (i.e., if it is a positive or negative sample). Next, training is performed based on the generated samples.

### C. Details About Training and Testing Stages

In our experiments, we use "leave-one-out" **cross validation**, which is a standard procedure to ensure separation of testing and training data sets. Specifically, assuming we have $N$ designs, to test each design, we use the $N-1$ remaining designs for training the model. This means 1) for a specific design/split layer, samples used in the training and testing processes do not overlap, and 2) different models are built to test different designs/split layers. Note that we use this cross validation process simply because we only have limited number of designs available. In practice, it is not required that the attacker use $N-1$ designs for training. For example, the training samples can be extracted from a fixed number of designs, which may come from reverse engineering, or historical data of complete/trial tape-outs of other designs with the same process technology.

For **training**, in our previous work [18], we used `RandomForest` in Weka [1] as the default classifier for its best performance among all classifiers we experimented. Random Forest is generally a robust and versatile model that has few hyperparameters, which works well on many classification and regression tasks in different fields, e.g. [6], [15]. Random Forest falls into the category of Bootstrap aggregating (Bagging) method, which combines the output of

a group of *base classifiers* to generate more robust predictions than individual classifiers. In the case of Random Forest, the base classifiers are decision trees. For a decision tree with $J$ decision nodes, its final prediction result can be derived as a Boolean combination of $J$ comparisons in the form of $x_i \leq t_j$, where $x_i$ is the value of the $i$-th feature and $t_j \in \mathbb{R}$ is the threshold in the $j$-th decision node. Since such comparisons and Boolean operations are nonlinear, it is powerful to deal with data that are not linearly separable, and is insensitive to outliers in the data, which exist in most machine learning problems, including the problem in this paper (See Figure 8).

In Weka, the base classifier for `RandomForest` is `RandomTree`, i.e., the randomized decision tree without pruning. Despite its good performance of classification, the runtime for training and testing is much longer than other classifiers.

In this paper, we keep using the Bagging method (with details in Section III-F). However, instead of using `RandomTree` as the base classifier, we use `REPTree` (Reduced Error Pruning tree), which prunes the tree where branching does not improve the performance on a validation data set. With this change, the runtime can be reduced in two aspects. First, `REPTree` has a smaller size than its corresponding `RandomTree` in general. Second, `REPTree` has better generalization performance in general owing to the pruning, which means the number of trees for the Bagging method can be reduced while maintaining similar performance as `RandomForest`. Both aspects translate to less training and inference time. Specifically, in Weka [1], the default number of `RandomTree`s in `RandomForest` is 100, while the default `REPTree`s in Bagging method is 10. Our experiments shows that Bagging with 10 `REPTree`s can achieve similar performance of classification to that of `RandomForest` with significant less runtime. Such comparison will be shown in Section IV-C.

Once the training is done, in order to do the **testing**, first the set of all possible unique v-pin pairs are considered for evaluation. Specifically, for each v-pin, all other v-pins in that split layer are considered for evaluation; if $n$ v-pins exist on the split layer, up to $\binom{n}{2}$ pairs are evaluated[2]. The testing stage then solves the inference problem by generating a yes/no answer for each pair to predict if the two v-pins are a match.

Once the testing stage is finished, the matches (i.e., yes answers) for each v-pin are bundled together into one set and recorded as its identified "List of Candidates" which we denote by **LoC**. We measure the quality of our modeling with two metrics. First, we measure the size of the LoC. Clearly, smaller-sized list is better. Second, we care about the classification accuracy, meaning that the LoC should include the actual match for the v-pin with a high likelihood.

### *D. Improvements for Scalability*

We faced **three issues** when applying our machine learning framework (denoted by ML) to the lower split layers. So we propose improvements to extend our framework to the lower layers. We denote this improved variation by Imp.

[2]Note, pairs connecting two output pins are disregarded.

**First**, we observed significant increase in the runtime of ML when applying it to lower split layers. This increase was both in the training and testing times (e.g., from seconds to hours when moving from layer 8 to layer 4). This was because there were a significantly larger number of samples generated for training, and more pairs to evaluate during testing. For training, the positive samples increased because there were a much larger number of v-pins located on the lower layers. (The number of negative samples also increased because it should be equal to that of positive samples.) The testing also took significantly longer runtime because the number of possible v-pin pairs to evaluate was quadratically higher. As the split layer goes lower, the runtime can be prohibitively long. This issue requires implementing improvements for scalability.

**Second**, when moving to lower layers, the classification accuracy of ML dropped (for example, from on-average 100.00% to 79.14% when moving from layer 8 to 6). We observed this was due to having "useless" negative samples during training. Recall the negative samples were selected by randomly picking two v-pins which were not a match. However, often times we observed the v-pins in negative samples were so far apart from each other on the split layer that their extracted layout features did not provide useful information for learning. In other words, just looking at the `ManhattanVpin` was sufficient to predict the pair was not a match. Also the remaining layout features may have acted as unwanted noise and degraded the learning.

**Finally**, we observed that moving to the lower layers resulted in the size of the generated LoC to grow at a much higher rate. Recall the size of LoC designates the number of candidates identified by ML as match for each v-pin. For a more effective attack we aim to minimize the size of LoC.

**To address** the above three issues, our main observation is to eliminate the useless negative samples during training. (See the second issue for explanation of useless samples.) Also, we aim to eliminate useless v-pin pairs from testing—they can sometimes degrade the proximity attack by generating false positives and increase the size of LoC. Specifically, instead of randomly picking negative samples during training and considering all v-pin pairs on the split layer during testing, we aim to consider a "useful" subset for each v-pin which are at a closer proximity to it. We explain how such a proximity is modeled shortly. The idea is that once a "neighborhood" is found, the samples for training (both positive *and* negative) and testing are only taken from the neighborhood which is then used to generate the LoC.

In general, we aim for the neighborhood size to be significantly smaller than the overall area of the split layer to reduce the number of tested v-pin pairs as much as possible, thus improve runtime, and focus on useful samples so that the accuracy is not impacted much. We note prior work [5] also focused on identifying a small neighborhood around each v-pin using linear regression, and declaring *all* v-pins in the neighborhood in the LoC. Compared to that, our neighborhood size will be much larger and while Imp considers all the v-pins in the neighborhood, it ultimately selects a subset of the v-pins to be included in the LoC.

Now we explain how the neighborhood is identified. As mentioned before, we use leave-one-out cross valida-

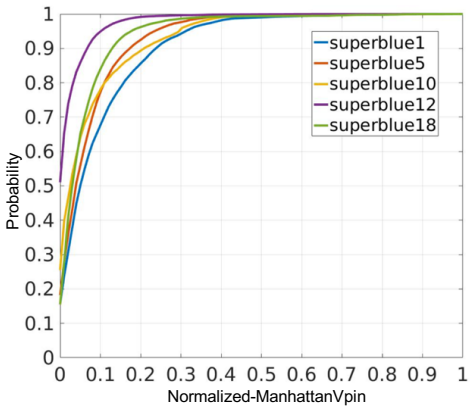


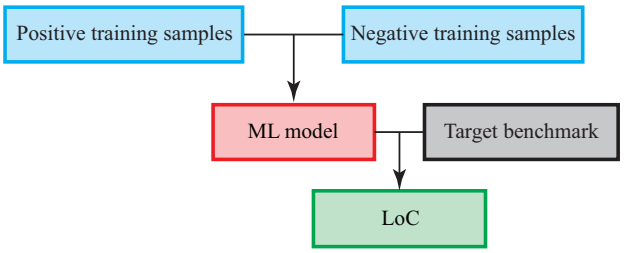

Fig. 4. Cumulative distribution function of the (normalized) `ManhattanVpin` feature is shown for the truly-matching v-pins in the training data set for each design.

tion for testing and training. To determine the neighborhood for one design, we first study the distribution of the `ManhattanVpin` feature in the remaining $N-1$ designs for the actual v-pin pairs (that are true match). Specifically, for each v-pin in the training data set, we already know its match and can measure the Manhattan distance between the two. This is done for all v-pins in the $N-1$ designs. Next, the neighborhood size is determined such that around 90% of all these v-pin pairs are included in it. The Imp variation uses this neighborhood for training by only generating samples for the v-pins included in it.

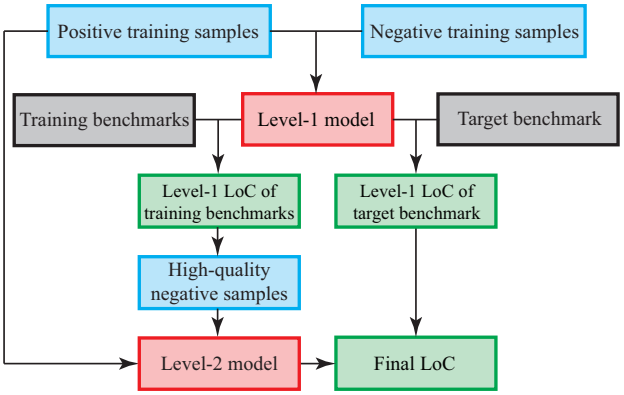
Fig. 4 shows the Cumulative Distribution Function (CDF) for different designs for layer 6 as the split layer. The curve for each design represents the aggregate data of the remaining $N-1$ designs. We emphasize that this process ensures the testing and training data are separated. Moreover, this step is done significantly faster than the testing and training because the number of actual v-pin pairs (that are a true match) are significantly smaller than all considered candidates. Finally, we note the 90% cutting point can be changed in order to trade off the runtime of Imp with its classification accuracy. Defining the neighborhood based on a smaller percentage, say 80%, can accelerate training and testing, however it excludes (from the training process) the knowledge of those 20% of connected v-pins which are further apart. So the classification accuracy may slightly degrade during the testing stage. On the other hand, with this modification, the aforementioned scalability issues can be relieved, so that this framework is feasible for use in large, real-world designs.

### *E. Improvements for Result Quality with Two-level Pruning*

From our experiments, we have found that the quality of negative samples plays an important role in training good machine learning models and achieving better final results. Thus, to further improve our results, we would want to pick negative samples with higher quality than those chose by the previous neighborhood sampling method. Moreover, we also noticed that the accuracies in previous experiments are almost perfect or relative high, which suggests the ground truth matching v-pin is included in the LoC with very high probability. At the same time, the other v-pins in the LoC are




(a) No pruning.




(b) Two-level pruning.

Fig. 5. Illustration of two-level pruning, compared with no pruning applied.

the set of v-pins that cannot be correctly distinguished by our previous Imp model. Therefore, they can be treated as "high-quality" negative samples. Thus, our idea of improving the results is to train a Level-2 machine learning model on top of the results generated by the Level-1 model, and perform Level-2 classifications only for those v-pins in the Level-1 LoC, with the intention that the Level-2 model focuses more on distinguishing v-pins that cannot be distinguished by the Level-1 model. Fig. 5(b) illustrates how the proposed two-level pruning works.

In order to correctly perform the two-level pruning procedure with leave-one-out cross-validation, some extra caution needs to be taken. Specifically, the LoCs generated from the previous experiments cannot be directly used in building the Level-2 model, otherwise the cross-validation procedure will be violated. The correct procedure is described as follows. Suppose there are $N$ benchmarks, one benchmark (referred to as *target benchmark*) will be held out for final testing, and the remaining $N-1$ benchmarks will be used for building the two-level pruning machine learning model. We build the Level-1 model using these $N-1$ benchmarks just like what we did before. Then we test the $N-1$ training benchmarks with this model to generate a Level-1 LoC for each v-pin in the training benchmarks. Then for each v-pin in the $N-1$ training benchmarks, we randomly pick a non-matching v-pin from its Level-1 LoC to form a high-quality negative sample. We then use these high-quality negative samples along with all positive samples to train the Level-2 model. Finally, we test this two-level machine learning model with the target benchmark in two steps: **1)** use the Level-1 model to generate the Level-1 LoC of the target benchmark; **2)** apply the Level-2 model on this Level-1 LoC to get the final LoC. The results of two-level pruning will be shown in Section IV-D.

### F. Controlling LoC Sizes

So far, all v-pin pairs under test will be assigned a binary yes/no answer indicating whether they are connected or not, resulting a LoC for each v-pin and a single value of accuracy. The problem of this approach appears when comparing models with different configurations (e.g., number of features, whether to apply bounding box, whether to apply two-level pruning) or with models in prior work. For example, one model may generate a larger (smaller) average size of LoC with a higher (lower) accuracy than the other. In this case we cannot tell directly which model is better. Also, we cannot answer questions like "what is the accuracy if we consider a LoC that contains 1% of the v-pins" or "what is the average size of LoC if we want an accuracy of 90%." Therefore, we need a method to control the LoC size.

Recall that Bagging, which we use in this paper, is a meta-classifier that combines the results of a group of base classifiers. There are several ways of such combination. Examples are 1) *hard voting* (a.k.a. majority voting), where each base classifier outputs a binary result and the combiner takes the majority of them, and 2) *soft voting*, where each individual classifier outputs a probability that the sample is positive, and the combiner takes the average of them. In the Weka implementation of Bagging, the soft voting method is adopted. Specifically, with `REPTree` as the base classifier, for a specific v-pin pair $(v, v')$ in the inference stage, the output probability $p_i(v, v')$ of tree $i$ is defined as

$$p_i(v, v') = P_i(v, v')/(P_i(v, v') + N_i(v, v')), \tag{1}$$

where $P_i(v, v')$ and $N_i(v, v')$ are the numbers of positive and negative samples in the training set that fall into the same leaf node as the pair $(v, v')$ does. With $n$ trees in total, the final output for pair $(v, v')$ is a binary value $r(v, v')$ given by

$$r(v, v') = \begin{cases} 1, & \text{if } p(v, v') \geq t \equiv 0.5, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where

$$p(v, v') = \sum_{i=1}^{n} p_i(v, v')/n. \tag{3}$$

Note that in (2), a default threshold of $0.5$ is applied in binary classification. To control the LoC size, in this paper, we generalize this bagging classifier by varying the threshold $t$ in (2), so that we can obtain a series of measurements of average LoC size and accuracy corresponding to different thresholds. Specifically, instead of outputting binary answers (yes/no), we record $p(v, v')$ for each tested v-pin pair $(v, v')$. Then the user can specify the desired threshold or LoC size to derive the LoCs without re-running the entire classification process. With this modification, we can obtain a more comprehensive behavior of the model showing the trade-off between LoC size and accuracy. We can also conduct meaningful comparison between different model configurations, as will be shown in Section IV.

### G. Limiting DiffVpinX or DiffVpinY in Highest Via Layer

In real CMOS VLSI layouts, most metal wires in the same level are routed in the same direction (i.e., either horizontally or vertically). This property facilitates the attack when the split layer is the highest via layer, since matching v-pin pairs in the top metal layer (i.e. the only layer above the split layer) must have an $x$- (or $y$-) distance of zero if that layer is vertically (or horizontally) routed. Therefore, for the attack in the highest via layer, we impose the limit on `DiffVpinX` or `DiffVpinY` to zero (depending on whether the top metal layer is vertically or horizontally routed) when generating the training set, and ignore (i.e., classify as disconnected) "ineligible" pairs when testing. Note that this modification does not work for lower split layers, because matching pairs in lower via layers can be connected by both horizontal and vertical wire segments in higher metal layers. Thus, they may have both non-zero `DiffVpinX` and non-zero `DiffVpinY`. The results of such modification will be shown in Section IV-E.

### H. Proximity Attack

Proximity attack (PA) is a task defined as follows. Given a v-pin $v$ (referred to as "target v-pin") in the benchmark, PA matches $v$ with the v-pin in the LoC of $v$ with the smallest Manhattan distance $d$, formally,

$$v_{PA}(v) = \underset{v' \in \mathrm{LoC}(v)}{\operatorname{argmin}} \; d(v, v') \tag{4}$$

where $d(v, v')$ is the Manhattan distance of v-pin pair $(v, v')$. In case more than one candidate v-pin in LoC have the same $d(v, v')$, we pick the one with the highest $p(v, v')$, and if they tie again, we randomly pick one of them. The PA is successful if and only if $v$ and $v_{PA}(v)$ are actually connected.

Note that the result of PA may vary with the size of LoC, which in turn is a function of the threshold $t$. In other words, with the ability to control the LoC size, we gain an opportunity to improve the results of PA. However, if we apply a single threshold $t$ to different target v-pins, as we did in Section III-F, the resulting LoC may be empty for some target v-pins and very long for others. But in PA, for each target v-pin, we must pick exactly one candidate v-pin connected to it. Therefore, for the task of PA, it is more natural to control the size of LoC individually by applying different thresholds $t(v)$ for different target v-pins $v$. To avoid confusion, we will use PA-LoC to denote the LoC used specially for the purposed of PA.

To achieve good PA performance, the PA-LoC cannot be either too long or too short—if the PA-LoC is too short, it may not include the matching v-pin; if the PA-LoC is too long, a non-matching v-pin may be selected in PA because the PA-LoC contains a v-pin that has even smaller Manhattan distance than the matching v-pin: either way makes the PA fail. For each target v-pin, if we classify all the tested v-pins by $p(v, v')$ and $d(v, v')$ as shown in Fig. 6, in which $S_i$'s represent the sets of v-pins within certain $p$ and $d$ ranges, the following observations can be made.

1) The matching v-pin is in $S_0$, not in any of $S_1, \ldots, S_8$.
2) If $|S_4| + |S_6| + |S_7| > 0$, the PA will fail for the target v-pin regardless of the size of PA-LoC, because there are v-pin pairs with higher $p$ and shorter $d$ than those formed by the matching v-pin.
3) If the PA-LoC contains any v-pin in $S_1$, the PA will fail for the target v-pin. Because a v-pin in $S_1$, which is not

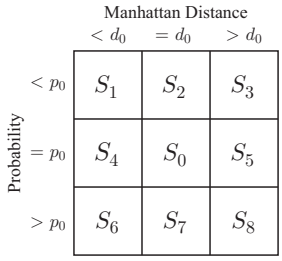



Fig. 6. Sets of v-pins around target v-pin $v$, classified by probability of connection and Manhattan distance. $p_0 = p(v, v_0)$ and $d_0 = d(v, v_0)$ are respectively the probability of connection (as evaluated by the classifier) and Manhattan distance of $v$ and $v_0$, where $v_0$ is the matching v-pin of $v$.

a match, will be selected in PA due to its shorter $d$ to the target v-pin.

4) If the PA-LoC does not contain all v-pins in $S_8$, the PA will fail for the target v-pin. Because the PA-LoC cannot contain the matching v-pin, which has a smaller $p$ than any v-pin in $S_8$.

Based on the above observations, we may perform a success PA only if $|S_4|+|S_6|+|S_7| = 0$. To maximize the success rate in such condition, the PA-LoC size of each individual target v-pin should be determined such that the PA-LoC contains all v-pins in $S_8$, but no v-pin in $S_1$. However, such optimal selection of PA-LoC size is not achievable for the target benchmark. Because a grid like Fig. 6 cannot be defined for them due to the lack of $d_0$ and $p_0$, which are the distance and probability of the pair composed by the target v-pin and its matching v-pin. Therefore, to determine a good PA-LoC size for the target benchmark, we go through a validation process as follows.

For each of the $N-1$ benchmarks used to train the ML model, we randomly selected 80% of v-pins. Then we mix the selected v-pins in these $N-1$ benchmarks to generate the training set as we did previously. All the other v-pins in these $N-1$ benchmarks are used for validation. We perform PA with different *PA-LoC fractions*, defined as the PA-LoC size divided by the total number of v-pins in the benchmark, to account for different numbers of v-pins in different benchmarks used in validation. The PA-LoC fraction that results in the best PA success rate in the validation process (averaged over $N-1$ benchmarks) will be used in the actual PA run.

### I. Design Obfuscation

Design obfuscation is an effective method to make the attack more difficult, as discussed in papers including [5], [8], [14], [16]. One effective approach is to increase the congestion so that the router is forced to use alternative, less straightforward routes.

To demonstrate how such obfuscation can impact the performance of attack in our framework, we manually add small random noises to the $y$-coordinate of all v-pins to mimic the effect of obfuscated routing, and apply the same approach to the new training and testing data sets. Detailed experiments and results can be found in Section IV-G.

## IV. Experimental Results

We experimented with the same setup as [5][3]. The designs were from the ISPD-2011 benchmark suite [12] with 9 metal layers and 8 via layers. We implemented the machine learning framework described in Fig. 1 which first generated a challenge case for a given split layer, extracted layout features, prepared the samples, applied training and then testing using the leave-one-out cross validation (See the beginning of Section III-C) to ensure separation between the testing and training data sets.

We used Weka [1] for training and specifically used Bagging method with base classifier `REPTree` and the default configuration. For each benchmark, we report the size of list of candidates (**LoC**) and accuracy with different thresholds applied. (See the end of Section III-B on how LoC is determined.) We also evaluated the success rate of proximity attack, as described in Section III-H. All experiments ran on an Intel Xeon X5670 CPU with Ubuntu 18.04 LTS and 24 GB memory.

In this paper, we present four configurations of our machine learning model as follows.

- **ML-9**: This is the configuration without improving the scalability (Section III-B), using the first 9 features introduced in Section III-B. This is the same configuration as "ML" in [18].
- **Imp-9**: This is the same as ML-9 except that we apply the technique in Section III-B to improve the scalability.
- **Imp-7**: This is the same with Imp-9 except that we excluded the two least important features (`TotalWireLength` and `TotalCellArea`) as shown in Section IV-A. This is the same configuration as "ML-Imp" in [18].
- **Imp-11**: This is the same with Imp-9 except that we use all 11 features introduced in Section III-A.

### A. Analyses of Feature Ranking and Data Distribution

Based on the training samples, we measure three statistical metrics, the correlation coefficient, the information gain, and the Fisher's discriminant ratio, to analyze the importance and class separability of 11 layout features defined in Section III-B. Correlation coefficient is a number that quantifies correlation between two variables. Information gain is the measure of reduction in the entropy of a variable achieved by learning the state of another one. Fisher's discriminant ratio measures how well the data of different classes can be separated. Specifically, we report 1) the information gain of a feature with respect to the output label, 2) the absolute value of correlation coefficient between a feature and the output label, and 3) the Fisher's discriminant ratio of each feature. For the first two metrics, a larger value means the feature is more important, and for the third metric, a larger value means matching and non-matching v-pin pairs are more separable in terms of this feature. For each feature, the three metrics are calculated based on the v-pin pairs which are used as samples for training "Imp" models. The correlation coefficient and information gain are measured using Weka [1], and the Fisher's discriminant ratio is calculated according to [10].

[3]Available at `http://homepages.cae.wisc.edu/~adavoodi`.

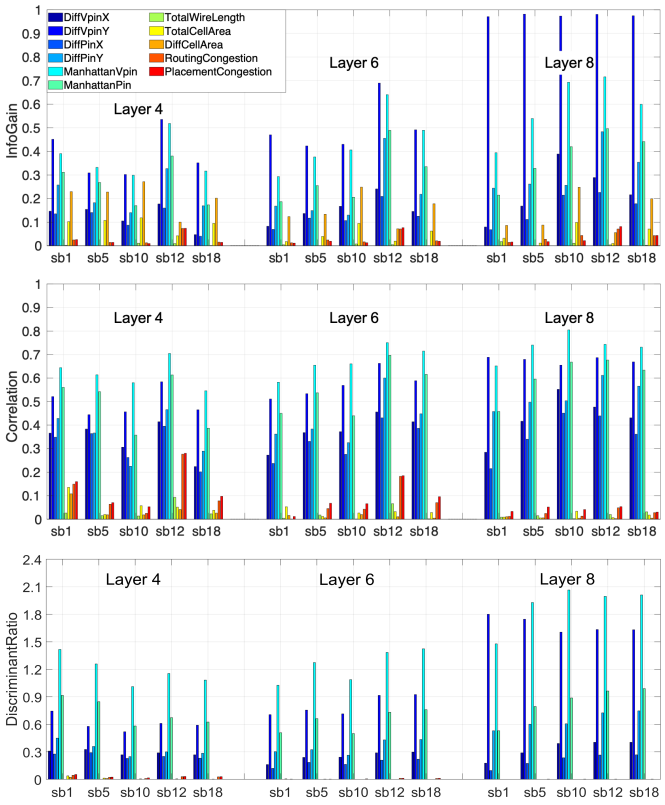



Fig. 7. Comparison of relative ranking of the 11 layout features using information gain, correlation coefficient, and Fisher's discriminant ratio.

The first two rows in Fig. 7 show the two importance metrics of features in each design for layers 4, 6 and 8. We make the following observations:
(1) The most important features are those related to the locations of the v-pins and then the Manhattan distance between the v-pins. Next, the location and the Manhattan distance between the pins (at the placement layers) become important. It shows that routing is more important than placement in analyzing the security of split manufacturing. Difference in the load/driver cell areas (`diffCellArea`) is the next important feature in terms of information gain. However, in one benchmark `superblue10` for layers 4 and 6, this feature is the third rank and more important than the pin-based features.
(2) In general, for almost all the features, their correlation coefficient and information gain are decreased when going from layer 8 to lower layers. This behavior indicates that these features are not equally powerful when classifying the samples from lower layers compared to layer 8. Thus, the classifier trained for lower layers performed somewhat worse than layer 8, as will be observed from Table IV.
(3) Correlation coefficient and information gain of features change relative to each other with change in layer; some features that are highly dominant in layer 8 becoming less dominant when moving to lower layers. Thus, other features become relatively more important. Specifically, the information gain of feature `DiffVpinY` in layer 8 in much higher than other features. Because there is no vertical (i.e. $y$-direction) routing layer above layer 8, all matching v-pin pairs have zero `DiffVpinY`—this property contains much information for classification, hence the high information gain. However, this property does not hold for lower layers. Since layers 4 and 6 are not the highest via layers, matching v-pins may be connected in $y$-direction in higher layers (e.g. layer 8) so they may have a non-zero `DiffVpinY`. That is why the information gains of `DiffVpinY` in layers 4 and 6 are not as high as that in layer 8.

The third row in Fig. 7 shows the Fisher's discriminant ratios of each feature in each design for layers 4, 6 and 8. It shows that location-related features about both v-pins and pins (`DiffVpinX`, `DiffVpinY`, `ManhattanVpin`, `DiffPinX`, `DiffPinY`, `ManhattanPin`) are much more powerful than other features to distinguish matching and non-matching v-pin pairs, especially for higher split layers. Among these features, `ManhattanVpin` is generally the most distinguishable feature, followed by `DiffVpinY` and `ManhattanPin`. This ranking is generally consistent with the ranking of feature importance discussed above.

Figure 8 shows the data distributions of each feature, separated by target classes. Due to the page limit, we only show the distributions for layer 6 as examples, where training data in all five benchmarks are mixed. Several observations can be made from this figure.
(1) Matching and non-matching v-pin pairs have overlapped distributions in all features. This means we cannot easily distinguish matching and non-matching v-pin pairs by examining any single feature.
(2) Some features (e.g. `ManhattanVpin`) show very different distributions for matching and non-matching v-pin pairs, which means these features are more powerful in distinguishing them. Some features (e.g. `PlacementCongestion`) show similar distributions for matching and non-matching v-pin pairs, which implies these features do not contribute much in our machine learning model.
(3) Due to the presence of special cells and macros in the benchmark, there are outliers in most features, especially in `TotalWireLength`, `TotalCellArea` and `DiffCellArea`. Since our `REPTree`-based machine learning model is insensible to outliers by nature, its performance is not negatively affected when these features are included, as we can see in Section IV-E.

### *B. Comparison of Effectiveness of the Attack with Prior Work*

We show in Table I the comparison between the results from the prior work [5] and our four settings (ML-9, Imp-9, Imp-7, Imp-11) for layers 4, 6, 8 as the split layers. For comparison with [5] we used the numbers reported in their paper and used an identical experimental setup. We report the following two metrics.

1) |**LoC**| designates the average size of the identified List of Candidates by each approach on each testing benchmark;
2) **Accuracy** measures the classification accuracy which is the percentage of the times that the actual match of a v-pin is included in its LoC;

Note that in [5] only one |LoC| and one accuracy is reported for each design. For fair comparisons, we control the size of

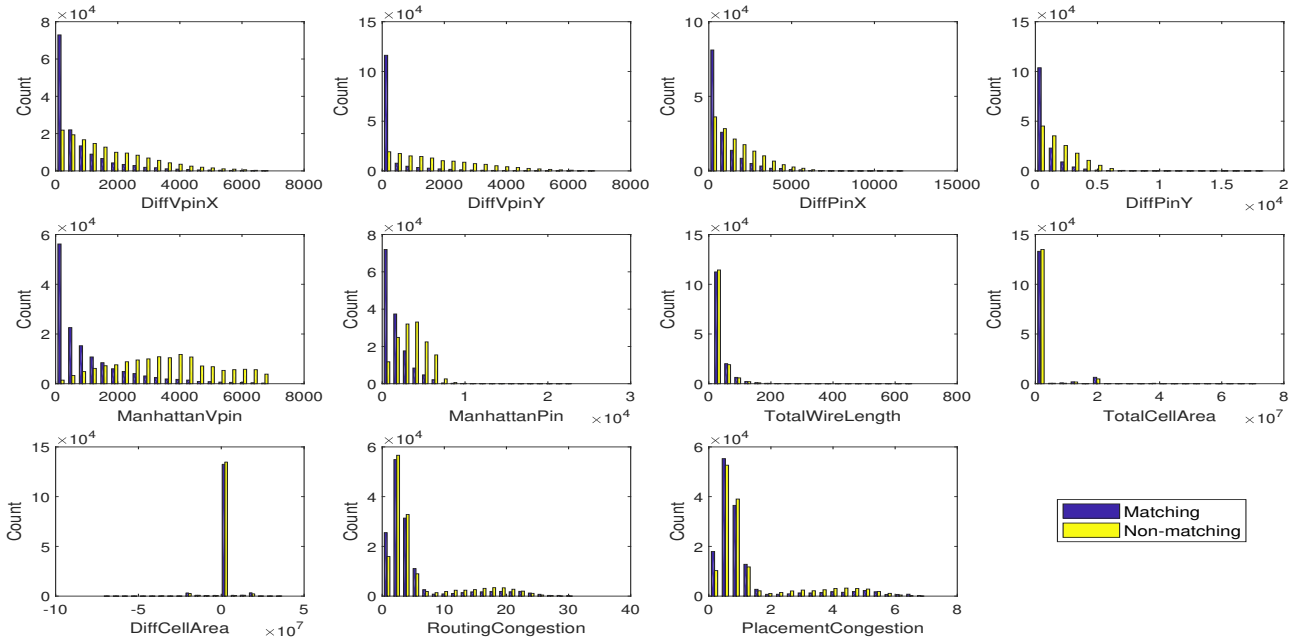



Fig. 8. Distributions of layout features in the training set for split layer 6.

TABLE I
COMPARISON OF OUR MACHINE LEARNING BASED APPROACHES WITH PRIOR WORK FOR DIFFERENT SPLIT LAYERS

| Split layer | Design Name | #v-pin | Prior Work [5] \|LoC\| | Prior Work [5] Accuracy | \|LoC\| with the same Accuracy as [5] ML-9 | Imp-9 | Imp-7 | Imp-11 | Accuracy with the same \|LoC\| as [5] ML-9 | Imp-9 | Imp-7 | Imp-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Layer 8 | superblue1 | 7824 | 115.1 | 15.53% | 1.8 | 1.5 | 2.2 | 2.6 | 100.00% | 99.95% | 99.95% | 99.95% |
| | superblue5 | 11018 | 149.4 | 35.63% | 5.0 | 3.9 | 3.7 | 3.7 | 100.00% | 100.00% | 100.00% | 100.00% |
| | superblue10 | 12888 | 185.4 | 42.45% | 1.6 | 0.9 | 1.5 | 2.7 | 100.00% | 100.00% | 100.00% | 100.00% |
| | superblue12 | 17312 | 870.4 | 73.13% | 17.6 | 20.3 | 30.0 | 15.0 | 100.00% | 100.00% | 100.00% | 100.00% |
| | superblue18 | 7518 | 280.7 | 66.88% | 9.5 | 9.9 | 8.1 | 7.0 | 100.00% | 100.00% | 100.00% | 100.00% |
| | **Avg** | 11312 | 320.2 | 42.72% | 7.1 | 7.3 | 9.1 | 6.2 | 100.00% | 99.99% | 99.99% | 99.99% |
| Layer 6 | superblue1 | 42998 | 487.8 | 33.40% | 11.3 | 9.6 | 10.7 | 9.4 | 73.34% | 74.13% | 74.14% | 74.81% |
| | superblue5 | 56173 | 506.8 | 39.40% | 22.4 | 17.6 | 19.7 | 16.1 | 77.14% | 77.60% | 77.15% | 79.31% |
| | superblue10 | 87212 | 687.9 | 64.03% | 213.3 | 198.0 | 198.1 | 186.9 | 80.15% | 81.37% | 81.52% | 81.95% |
| | superblue12 | 75994 | 2527.9 | 73.50% | 82.3 | 85.8 | 63.9 | 72.9 | 90.60% | 90.23% | 89.57% | 93.26% |
| | superblue18 | 33596 | 773.6 | 58.43% | 31.5 | 29.4 | 27.3 | 25.6 | 82.96% | 83.01% | 83.90% | 85.85% |
| | **Avg** | 59194 | 996.8 | 53.75% | 72.1 | 68.1 | 63.9 | 62.2 | 80.84% | 81.27% | 81.26% | 83.03% |
| Layer 4 | superblue1 | 149517 | 885.6 | 58.19% | 71.0 | 62.5 | 56.5 | 52.2 | 76.03% | 76.40% | 75.54% | 76.77% |
| | superblue5 | 178136 | 745.8 | 53.70% | 145.3 | 139.8 | 147.3 | 123.4 | 71.84% | 72.15% | 71.25% | 73.11% |
| | superblue10 | 215292 | 939.4 | 54.68% | 225.6 | 207.2 | 252.7 | 278.9 | 71.25% | 73.48% | 71.60% | 72.52% |
| | superblue12 | 170572 | 2078.8 | 75.67% | 443.9 | 459.4 | 584.8 | 296.7 | 87.56% | 87.50% | 85.03% | 89.80% |
| | superblue18 | 85146 | 1076.9 | 70.13% | 453.6 | 414.7 | 440.3 | 353.5 | 78.87% | 80.16% | 79.20% | 82.39% |
| | **Avg** | 159732 | 1145.3 | 62.47% | 267.9 | 256.7 | 296.3 | 220.9 | 77.11% | 77.94% | 76.52% | 78.92% |

LoC in our work such that either |LoC| or accuracy is aligned with that in [5], and then we compare the other metric of the two. As can be seen in Table I, for the same accuracy, the size of LoC in our work is much smaller of that in [5]; for the same LoC size, the accuracy in our work is always better than in [5]. Specifically, for layer 8, all four configurations provide similar results which are much better than [5]. For the same accuracy, the LoC size is less than 3% of the size in [5]. For the same LoC size, all configurations have similar accuracy close to 100%, compared to on-average 42.72% in [5]. Similar conclusions can be drawn for layers 6 and 4. Moreover, please note that testing and training data are not separated in [5].

### C. Results of Different Base Classifiers in Bagging

In this subsection, we show the efficacy of changing the base classifier of Bagging from `RandomTree` to `REPTree`. To this end, we use Imp-7 to compare the runtime and performance of attack in this paper with those in [18], where the corresponding model is referred to as "ML-Imp." We used the default settings in Weka [1] for both base classifiers. Again, since in [18] there is only one LoC size and accuracy reported for each benchmark, we compare by aligning one metric and compare the other, as we did for comparison with [5]. The results are shown in Table II. As shown in this table, the performance of attack is similar for both base classifiers. However, `REPTree`, as adopted in this paper, takes less than 10% of runtime compared to the counterpart in [18].

TABLE II
COMPARISON BETWEEN RANDOMTREE ( [18]) AND REPTREE (THIS PAPER) AS BASE CLASSIFIER WITH IMP-7

| Split layer | Design | [18] \|LoC\| | [18] Acc. | This paper \|LoC\| | This paper Acc. |
|---|---|---|---|---|---|
| Layer 8 | superblue1 | 15.3 | 99.82% | 15.2 | 99.85% |
| | superblue5 | 27.9 | 99.42% | 27.2 | 99.60% |
| | superblue10 | 20.8 | 100.00% | 19.4 | 100.00% |
| | superblue12 | 44.4 | 100.00% | 48.0 | 99.75% |
| | superblue18 | 23.1 | 99.97% | 23.3 | 99.87% |
| | **Avg** | 26.3 | 99.84% | 26.6 | 99.81% |
| | **Runtime** | | 7.25 min | | 0.48 min |
| Layer 6 | superblue1 | 553.4 | 74.65% | 555.1 | 74.64% |
| | superblue5 | 560.4 | 78.89% | 645.6 | 77.79% |
| | superblue10 | 740.8 | 82.56% | 759.4 | 82.30% |
| | superblue12 | 2648.7 | 90.18% | 2955.2 | 89.72% |
| | superblue18 | 793.1 | 83.40% | 716.8 | 84.08% |
| | **Avg** | 1059.3 | 81.94% | 1126.4 | 81.71% |
| | **Runtime** | | 10.73 hrs | | 0.42 hrs |

TABLE III
COMPARISON BETWEEN TWO-LEVEL PRUNING AND NO PRUNING WITH IMP-11

| Split layer | Design | Two-level pruning \|LoC\| | Two-level pruning Acc. | No pruning \|LoC\| | No pruning Acc. |
|---|---|---|---|---|---|
| Layer 8 | superblue1 | **3.15** | **40.95%** | 5.31 | 22.68% |
| | superblue5 | **4.33** | **57.51%** | 6.92 | 39.82% |
| | superblue10 | **4.54** | **79.87%** | 7.91 | 66.54% |
| | superblue12 | 8.73 | 38.49% | **5.40** | **60.01%** |
| | superblue18 | **5.46** | **67.86%** | 7.20 | 53.40% |
| | **Avg** | **5.24** | **56.94%** | 6.55 | 48.49% |
| | **Runtime** | | 111.7 sec | | 27.8 sec |

### D. Results of Two-level Pruning

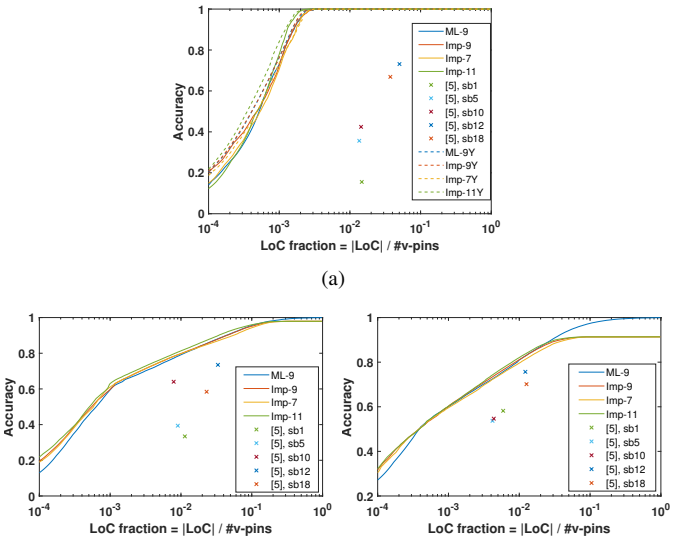
In this subsection, we show the efficacy of two-level pruning. To this end, we use Imp-11 to compare the performance of attack. Again, we compare by aligning one of |LoC| and accuracy and compare the other, as we did for comparison with [5]. The results are shown in Table III.

The column "Two-level pruning" in Table III shows the results of the two-level pruning procedure. Compared to the results where no pruning is applied, in layer 8, the performance is improved by virtue of two-level pruning in all benchmarks except for `superblue12`. We notice that `superblue12` has a different characteristics of v-pin distribution from that of the other four benchmarks, which may be the reason why two-level pruning works as desired for all but this benchmark.

When the split layer moves to layer 6, however, two-level pruning does not bring improvement for any benchmark compared to the results without pruning. As in layer 6, the accuracies of Level-1 model are not as high as those in layer 8, which worsens the Level-2 model, because Level-2 model depends on Level-1 LoC of training benchmarks, as described in Section III-E.

### E. Comparison of Model Configurations in This Work

In this work, we use four different configurations (ML-9, Imp-9, Imp-7, Imp-11) for split layers 4, 6, and 8, as well as four more configurations with suffix "Y" for layer 8 only. The configurations with "Imp" in the names improves the scalability as described in Section III-D. The numbers in the names of configuration indicate the numbers of features used in training and testing, as described at the beginning




(a)

(b) (c)

Fig. 9. The trade-off between LoC fraction and accuracy (averaged over five benchmarks) with the split layer being (a) layer 8, (b) layer 6, and (c) layer 4. Results in prior work [5] are plotted for comparison.

of Section IV. The four configurations suffixed "Y" impose limit on `DiffVpinY` in the training set, on top of the corresponding models without the suffix "Y", as described in Section III-G.

Figure 9 shows the trade-off between LoC fraction (defined as the LoC size divided by the number of v-pins in the corresponding benchmark) and accuracy in different ML model configurations. The horizontal axis shows the LoC fraction, which balances different numbers of v-pins in each benchmark. The vertical axis shows the average accuracy over five benchmarks. Higher curves indicate better performance of attack. For comparison, we also plot the results from [5] in this figure, so that the significant improvement of this work is clearly shown.

To quantitatively compare the results of different configurations, we can fix one coordinate in the figure and compare the other. For example, by drawing horizontal lines in Fig. 9, we can get the LoC fraction of each configurations under given accuracy values. Similarly, vertical lines give us the accuracy of each configuration under given LoC fractions. These values are shown in Table IV. We also include in Table IV the runtime of each configuration, averaged over five benchmarks.

From Table IV, we can make observations about the performance of attack and the runtime in the following aspects.

*1) Different split layers:* We first compare the same configuration applied in different split layers. In layer 8, we can get an accuracy of 95% with only 0.2% of v-pins in the benchmark as candidate on average, with 1% of v-pins as candidate, we can achieve near 100% accuracy. The performance becomes worse as the split layer goes down to layers 6 and 4. Specifically, with 1% of v-pins as candidate, the accuracy drop from near 100% to around 80% when the split layer goes from layer 8 down to layers 6 and 4. And if we want an accuracy of 95% in layers 6 or 4, we need to include around 10% of all v-pins in the candidate list. Comparing the results for layers 6 and 4, we can see that although the LoC fraction are similar

TABLE IV
COMPARISON OF DIFFERENT MACHINE LEARNING MODEL CONFIGURATIONS IN THIS WORK

| Split layer | Configuration | LoC fraction with an average accuracy of ... 95% | 90% | 80% | 50% | Average accuracy with a LoC fraction of ... 0.01% | 0.1% | 1% | 10% | Runtime |
|---|---|---|---|---|---|---|---|---|---|---|
| Layer 8 | ML-9 | 0.20% | 0.18% | 0.13% | 0.05% | 14.11% | 73.03% | 100.00% | 100.00% | 33.6 sec |
| | Imp-9 | 0.20% | 0.18% | 0.13% | 0.05% | 20.10% | 72.46% | 99.99% | 99.99% | 30.6 sec |
| | Imp-7 | 0.20% | 0.18% | 0.14% | 0.05% | 14.92% | 70.68% | 99.99% | 99.99% | 28.8 sec |
| | Imp-11 | 0.18% | 0.16% | 0.11% | 0.05% | 12.28% | 77.93% | 99.99% | 99.99% | 27.8 sec |
| | ML-9**Y** | 0.19% | 0.17% | 0.12% | 0.04% | 21.68% | 76.85% | 99.99% | 100.00% | 13.9 sec |
| | Imp-9**Y** | 0.18% | 0.16% | 0.11% | 0.04% | 21.21% | 76.03% | 99.99% | 99.99% | 13.9 sec |
| | Imp-7**Y** | 0.19% | 0.16% | 0.12% | 0.05% | 20.60% | 75.08% | 99.99% | 99.99% | 13.1 sec |
| | Imp-11**Y** | 0.16% | 0.14% | 0.10% | 0.04% | 22.19% | 80.09% | 99.99% | 99.99% | 16.6 sec |
| Layer 6 | ML-9 | 9.07% | 4.35% | 1.12% | 0.06% | 13.01% | 59.51% | 79.14% | 95.57% | 45.1 min |
| | Imp-9 | 9.72% | 4.46% | 1.03% | 0.05% | 19.52% | 59.72% | 79.80% | 95.13% | 22.9 min |
| | Imp-7 | 11.25% | 5.46% | 1.04% | 0.06% | 18.93% | 61.06% | 79.76% | 94.28% | 24.9 min |
| | Imp-11 | 8.00% | 3.40% | 0.83% | 0.05% | 21.83% | 63.02% | 81.34% | 95.96% | 19.0 min |
| Layer 4 | ML-9 | 5.75% | 2.86% | 0.93% | 0.04% | 27.16% | 60.03% | 80.98% | 97.40% | 5.31 hrs |
| | Imp-9 | — | 3.82% | 0.88% | 0.04% | 31.76% | 60.16% | 81.09% | 91.32% | 0.96 hrs |
| | Imp-7 | — | 4.64% | 1.05% | 0.04% | 30.53% | 59.56% | 79.60% | 91.32% | 1.06 hrs |
| | Imp-11 | — | 3.27% | 0.79% | 0.04% | 32.15% | 60.32% | 82.08% | 91.34% | 0.92 hrs |

for a given accuracy, as the absolute number of v-pins are several times larger in layer 4 than in layer 6, it is generally harder for the attacker to reverse engineer given partial layout at split layer 4 than layer 6.

The runtime of attack becomes longer from seconds to hours as the split layer goes from layer 8 down to layer 4, especially for the configuration without improved scalability (i.e. ML-9), due to the increasing number of v-pins in lower layers. From the designers' perspective, it means lower split layers generally provide more security, which agrees with the intuition that a lower split layer implies less information available to the attacker. However, this observation is simply based on the machine learning approach and the extracted features described in this paper. In reality, which split layer is secure enough remains an open question, which depends on other knowledge and techniques available to the attacker to further refine the results.

*2) With or without improved scalability:* Comparing Imp-9 with ML-9 in Table IV, we see similar performance of attack in terms of LoC fraction and accuracy with runtime speedups of $1.1\times$–$5.5\times$ depending on the split layer. Note that the speedup is more significant as the split layer goes down, which shows the improved configuration (Imp-9) does have better scalability than the baseline (ML-9). From Fig. 9(b) and (c), we can also see a flat, saturated portion on the right side of Imp-9, Imp-7 and Imp-11 curves. This is due to the fact that v-pin pairs that are too far apart were completely ignored in the testing stage. Thus, some truly connected v-pin pairs that are far apart can never be included in the LoC, regardless of the LoC size. This conforms with the description in Section III-D. Please note in layer 4, the accuracies saturate at around 91% in improved configurations for this reason. Therefore they cannot reach an average accuracy of 95% with any LoC fraction (shown as dashes in Table IV). As mentioned in Section III-D, this saturated accuracy will be higher if a larger neighborhood is applied, at the cost of longer runtime for testing more v-pin pairs, and vice versa.

*3) Different numbers of features:* In Fig. 9, especially Fig. 9(b), by comparing the configurations with different numbers of features (Imp-9, Imp-7 and Imp-11, where 9, 7, and 11 features are involved in the training and testing set, respectively), we can see that, in general, Imp-9 is slightly better than Imp-7, but is slightly worse than Imp-11 in terms of performance of attack. This observation aligns with the fact that the 7 features used in Imp-7 include most important information needed for classification, as indicated in Section IV-A. If we add some features of less importance (as in Imp-9 and Imp-11), the performance can be slightly improved, but not too much. The runtime does not vary significantly with the number of features, either. Note that the two features added from Imp-7 to Imp-9 are `TotalWireLength` and `TotalCellArea`, which have some extreme values in the dataset according to the distribution in Fig. 8. Since the `REPTree`-based classifier is not sensitive to outliers by nature, including these features does not negatively affect the performance.

*4) With or without limiting* `DiffVpinY` *for the highest via layer:* For layer 8 (the highest via layer), comparing the configurations with limiting `DiffVpinY` (the four configurations with suffix "Y") with the ones without such limit, we can observe an improvement on performance in every configuration, which is shown in both Table IV and Fig. 9(a). For example, with 0.1% of v-pins included in LoC (i.e. $\approx 10$ candidates), we achieve 76.03% accuracy on average with Imp-9Y, compared to 72.46% with Imp-9. The runtime is also reduced approximately by half with such limit, since most v-pin pairs are ignored without actual testing due to their non-zero `DiffVpinY`. From the designers' perspective, it confirms that splitting at the highest via layer is insecure and should be avoided in practice.

### *F. Results of Proximity Attack*

Table V shows the success rate of PA and the runtime overhead for the validation process in PA (see Section III-H).

It is shown that the proposed PA method achieves up to $11\times$ higher success rate than that in [5] for the only reported benchmark `superblue1`. Besides, several other observations can be made from Table V.

**First**, model configurations with improved scalability (Imp-9, Imp-7 and Imp-11, as described in Section III-D) or limits on `DiffVpinY` for the highest via layer (ML-9Y, Imp-9Y, Imp-7Y and Imp-11Y, as described in Section III-G) generally have better PA success rate than ML-9, since these

TABLE V
PROXIMITY ATTACK SUCCESS RATE AND RUNTIME OF CROSS VALIDATION FOR DIFFERENT MODEL CONFIGURATIONS

| Design | %PA: Prior Work | | %PA from Cross Validation | | | |
|---|---|---|---|---|---|---|
| | [5] | [18] | ML-9 | Imp-9 | Imp-7 | Imp-11 |
| **Split layer 8:** | | | | | | |
| sb1 | 1.95% | 11.26% | 15.21% | 14.84% | 13.03% | 11.05% |
| sb5 | — | 21.20% | 20.04% | 21.22% | 21.35% | 21.69% |
| sb10 | — | 60.72% | 42.97% | 59.54% | 57.78% | 42.30% |
| sb12 | — | 11.72% | 10.96% | 15.03% | 13.84% | 11.53% |
| sb18 | — | 18.26% | 13.41% | 17.56% | 18.43% | 17.85% |
| **Avg** | — | 24.63% | 20.52% | 25.64% | 24.89% | 20.88% |
| **Time** | — | — | 91.4 sec | 101.8 sec | 59.0 sec | 60.0 sec |
| **Split layer 6:** | | | | | | |
| sb1 | 0.76% | 2.11% | 4.17% | 5.47% | 5.13% | 5.56% |
| sb5 | — | 2.28% | 4.06% | 5.07% | 5.57% | 6.19% |
| sb10 | — | 5.82% | 6.72% | 7.62% | 7.35% | 5.47% |
| sb12 | — | 3.40% | 3.79% | 4.63% | 6.23% | 5.75% |
| sb18 | — | 3.08% | 5.03% | 6.71% | 6.12% | 6.46% |
| **Avg** | — | 3.34% | 4.75% | 5.90% | 6.08% | 5.89% |
| **Time** | — | — | 34.2 min | 25.5 min | 23.3 min | 20.0 min |
| **Split layer 4:** | | | | | | |
| sb1 | 0.64% | 2.58% | 5.19% | 6.51% | 6.88% | 7.11% |
| sb5 | — | 1.61% | 3.00% | 4.36% | 4.44% | 4.63% |
| sb10 | — | 2.50% | 3.83% | 4.38% | 3.66% | 2.85% |
| sb12 | — | 3.48% | 3.84% | 5.03% | 4.76% | 4.86% |
| sb18 | — | 2.47% | 3.56% | 5.29% | 5.00% | 5.22% |
| **Avg** | — | 2.53% | 3.88% | 5.11% | 4.95% | 4.93% |
| **Time** | — | — | 3.27 hrs | 1.27 hrs | 1.27 hrs | 1.27hrs |
| **Design** | **[5]** | **[18]** | **ML-9Y** | **Imp-9Y** | **Imp-7Y** | **Imp-11Y** |
| **Split layer 8:** | | | | | | |
| sb1 | 1.95% | 11.26% | 19.06% | 18.56% | 17.66% | 17.99% |
| sb5 | — | 21.20% | 26.60% | 26.44% | 25.35% | 26.00% |
| sb10 | — | 60.72% | 57.88% | 58.72% | 52.32% | 46.38% |
| sb12 | — | 11.72% | 13.42% | 14.02% | 14.28% | 14.64% |
| sb18 | — | 18.26% | 23.33% | 21.36% | 21.08% | 22.26% |
| **Avg** | — | 24.63% | 28.06% | 27.82% | 26.14% | 25.45% |
| **Time** | — | — | 36.2 sec | 29.6 sec | 26.8 sec | 30.0 sec |

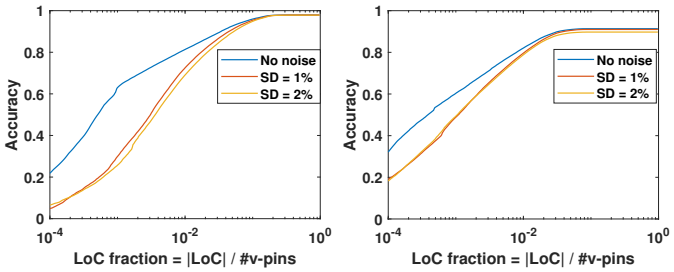
improved configurations filter out most "low-quality" candidates even before PA.

**Second**, we exploit the opportunity of improving the PA results by using a proper size of PA-LoC. When determining the PA-LoC, the validation technique described in Section III-H generally results in a higher success rate than applying a fixed threshold $t = 0.5$ in (2) as in [18], especially for layers 6 and 4.

**Third**, the effect of validation is suboptimal for benchmark `superblue10`, especially for layer 8, due to the difference in characteristics of this benchmark than those of the other four (from the fact that the PA success rate for `superblue10` is much larger than the other four). Moreover, these characteristics cannot be well captured in the validation process for `superblue10`, where the available information is only from the other four benchmarks. This issue becomes worse when the training set is not pruned for scalability (as in ML-9), or more features are involved (as in Imp-11).

**Finally**, the *extra* runtime for the validation-based PA is similar across configurations with different numbers of features. Such extra runtime is about 1–3× compared to the runtime of the machine learning procedure shown in Table IV.

Although the success rate of PA in Table V is not high enough, especially for layers 6 and 4, it is not sufficient to conclude if these layers should be selected as the split layer or not from the designers' perspective. We should note that the rates in Table V are the fractions that the attackers




(a) (b)

Fig. 10. The LoC fraction and accuracy (averaged over five benchmarks) with the split layer at (a) layer 6, and (b) layer 4, with and without added noises.

TABLE VI
PROXIMITY ATTACK SUCCESS RATE WITH AND WITHOUT DATA NOISES

| Split layer | Design | No noise | SD = 1% | SD = 2% |
|---|---|---|---|---|
| Layer 6 | sb1 | 5.56% | 1.39% | 1.46% |
| | sb5 | 6.19% | 1.05% | 0.95% |
| | sb10 | 5.47% | 1.20% | 0.80% |
| | sb12 | 5.75% | 0.89% | 1.07% |
| | sb18 | 6.46% | 1.52% | 1.40% |
| | **Avg** | 5.89% | 1.21% | 1.14% |
| Layer 4 | sb1 | 7.11% | 3.94% | 3.51% |
| | sb5 | 4.63% | 2.62% | 2.72% |
| | sb10 | 2.85% | 1.14% | 0.98% |
| | sb12 | 4.86% | 1.34% | 1.89% |
| | sb18 | 5.22% | 2.14% | 2.21% |
| | **Avg** | 4.93% | 2.24% | 2.26% |

successfully identify a **single** matching v-pin, **only** with the described machine learning approach and layout features. Even though machine learning does not always identify the matches directly, it still helps the attackers eliminate most v-pins that are unlikely to be matches, as discussed in the previous subsection. In reality, as mentioned earlier, the attackers may opt to obtain a larger LoC (which means higher accuracy), and apply other domain knowledge about the design and/or other observations to further refine the LoC and understand the design. As an intuitive example, if regular and repeated patterns are observed in the layout, they may be assumed to have similar logic function in the design (e.g. data bus connections). With extra knowledge like this, it may be easier for the attackers to reverse engineer successfully.

### G. Results of Design Obfuscation

To imitate the effect of design obfuscation as discussed in Section III-I, we add Gaussian white noises to the $y$-coordinate of all v-pins in layers 6 and 4 in each benchmark, respectively. We do not experiment on layer 8 because otherwise it requires both horizontal and vertical routing in the top metal layer (M9), which is usually impractical. The standard deviation of the noises equals 1%–2% of the layout size in $y$-direction. Such noises affect the two of the most important features: `DiffVpinY` and `ManhattanVpin`, according to the feature ranking in Section IV-A. Then we apply the same procedure of training and testing with configuration Imp-11 to observe how different the results would be with the noises.

The LoC ratio-vs-accuracy curves with and without noises are shown in Fig. 10. We can see that the added noises, though fairly small, significantly affect the performance of attack.

The vertical gap between the blue and red lines indicates the difference in accuracy with the same LoC ratio. Such accuracy difference can be up to 0.3 (e.g. a difference of 0.6 and 0.3 at the LoC ratio of $10^{-3}$ for layer 6). The horizontal gap indicates the difference in LoC ratios needed to guarantee the same accuracy. Such difference can be up to $5\times$ as measured in the figure for layer 6 at the accuracy around 60%. The added noises have smaller effects in layer 4 than in layer 6, because the variation of $y$-coordination in layer 4 is already larger than layer 6 before adding the noises, which makes the added noise relatively smaller and less effective.

The success rate of proximity attack with and without noises are shown in Table VI. As can be seen, the average success rate of proximity attack drops significantly (up to 81%) in layer 6 and mildly (up to 55%) in layer 4.

As suggested in both Fig. 10 and Table VI, a noise with standard deviation around 1% of the layout size is good enough for the purpose of obfuscation. Increasing the magnitude of noises further does not make much difference.

## V. Conclusions

We developed novel and scalable machine learning techniques to analyze the security of split manufacturing on large designs. We showed significantly better results compared to prior work using the same setup. We also analyzed ranking of various layout features taken from placement, routing, and cell libraries used in the netlists, and showed the routing features are in general the most helpful to the learning process. This emphasizes the importance of using a correct setup (in terms of number of metal layers, and proper control of the routing algorithm) in generating the challenge test cases to study split manufacturing. Our studies also showed various challenges we addressed to handle the scalability issue when the split layer is lower. It further emphasizes the importance of considering computational feasibility as a constraint dictated by the experimental environment when designing an attack algorithm.

## References


[1] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, 4th ed. Morgan Kaufmann, 2016.

[2] B. Hill, R. Karmazin, C. T. O. Otero, J. Tse, and R. Manohar, "A split-foundry asynchronous FPGA," in *Custom Integrated Circuits Conf.*, 2013, pp. 1–4.

[3] M. Jagasivamani, P. Gadfort, M. Sika, M. Bajura, and M. Fritze, "Split-fabrication obfuscation: Metrics and techniques," in *Proc. Int. Symp. on Hardware-Oriented Security and Trust*, 2014, pp. 7–12.

[4] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: A review," *GESTS Int. Trans. Computer Science and Engineering*, vol. 30, pp. 25–36, November 2005.

[5] J. Magaña, D. Shi, J. Melchert, and A. Davoodi, "Are proximity attacks a threat to the security of split manufacturing of integrated circuits?" *IEEE Trans. VLSI Syst.*, vol. 25, no. 12, pp. 3406–3419, 2017.

[6] J. Melchert, B. Zhang, and A. Davoodi, "A comparative study of local net modeling using machine learning," in *Proc. Great Lakes Symposium on VLSI*. New York, NY, USA: ACM, 2018, pp. 273–278.

[7] C. T. O. Otero, J. Tse, R. Karmazin, B. Hill, and R. Manohar, "Automatic obfuscated cell layout for trusted split-foundry design," in *Proc. Int. Symp. on Hardware-Oriented Security and Trust*, 2015, pp. 56–61.

[8] S. Patnaik, J. Knechtel, M. Ashraf, and O. Sinanoglu, "Concerted wire lifting: Enabling secure and cost-effective split manufacturing," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan 2018, pp. 251–258.

[9] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *Design, Autom. and Test in Europe*, 2013, pp. 1259–1264.

[10] J. S. Sánchez, R. A. Mollineda, and J. M. Sotoca, "An analysis of how training data complexity affects the nearest neighbor classifiers," *Pattern Anal. Appl.*, vol. 10, no. 3, pp. 189–201, Jul. 2007.

[11] A. Senguptay, S. Patnaiky, J. Knechtelz, M. Ashraf, and S. Garg, "Rethinking split manufacturing: An information-theoretic approach with secure layout techniques?" in *Proc. Int. Conf. Comput.-Aided Design*, 2017.

[12] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li, G.-J. Nam, and J. A. Roy, "The ISPD-2011 routability-driven placement contest and benchmark suite," in *Proc. Int. Symp. on Physical Design*, 2011, pp. 141–146.

[13] Y. Wang, P. Chen, J. Hu, and J. Rajendran, "The cat and mouse in split manufacturing," in *Design Autom. Conf.*, 2016, p. 165.

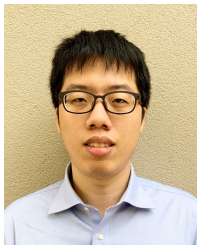
[14] ——, "Routing perturbation for enhanced security in split manufacturing," in *Proc. Asia South Pacific Design Autom. Conf.*, 2017, pp. 605–510.

[15] D. Wu, C. Jennings, J. Terpenny, R. X. Gao, and S. Kumara, "A comparative study on machine learning algorithms for smart manufacturing: Tool wear prediction using random forests," *Journal of Manufacturing Science and Engineering*, vol. 139, no. 7, pp. 1–9, 2017.

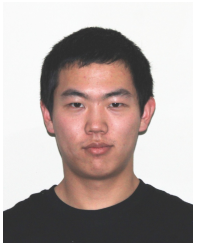
[16] K. Xiao, D. Forte, and M. M. Tehranipoor, "Efficient and secure split manufacturing via obfuscated built-in self-authentication," in *Proc. Int. Symp. on Hardware Oriented Security and Trust*, 2015, pp. 14–19.

[17] W. Xu, L. Feng, J. Rajendran, and J. Hu, "Layout recognition attacks on split manufacturing," in *Proc. Asia South Pacific Design Autom. Conf.*, 2019, pp. 45–50.

[18] B. Zhang, J. Magaña, and A. Davoodi, "Analysis of security of split manufacturing using machine learning," in *Proc. Design Autom. Conf.*, 2018, pp. 141:1–141:6.

**Wei Zeng** received the B.S. degree in microelectronics and the M.S. degree in microelectronics and solid-state electronics from Fudan University, Shanghai, China, in 2014 and 2017, respectively. He was a reciprocal exchange student in electrical engineering with the University of California at Davis, in 2012. He is currently pursuing the Ph.D. degree in computer engineering with the University of Wisconsin–Madison. His current research interests include machine learning on design and analysis of integrated circuits.



**Boyu Zhang** received the B.S. degree in electrical engineering from Miami University in 2015. He is currently pursuing the Ph.D. degree in computer engineering with the University of Wisconsin–Madison. His research interests include efficient and high-speed realization of deep neural networks on embedded systems, design space exploration of DNN architectures, and incorporation of machine learning algorithms in CAD tools.

**Azadeh Davoodi** (SM'13) is an Associate Professor of Electrical and Computer Engineering at the University of Wisconsin–Madison. Her primary research interests are in Electronic Design Automation and debug of Integrated Circuits, hardware security, and in Design Automation of Things, in general. Azadeh is recipient of a 2011 NSF CAREER award. Her work with collaborators received the best paper awards of the 2015 ACM Transactions on Design Automation of Electronic Systems and Best Paper nomination at the 2010 Design Automation Conference.