APPLE: An Explainer of ML Predictions on Circuit Layout at the Circuit-Element Level

Tao Zhang[1], Haoyu Yang[2], Kang Liu[3], Zhiyao Xie[1*]
[1]Hong Kong University of Science and Technology, [2]NVIDIA, [3]Huazhong University of Science and Technology
tao.zhang@connect.ust.hk, haoyuy@nvidia.com, kangliu@hust.edu.cn, eezhiyao@ust.hk

***Abstract*—In recent years, we have witnessed many excellent machine learning (ML) solutions targeting circuit layouts. These ML models provide fast predictions on various design objectives. However, almost all existing ML solutions have neglected the basic interpretability requirement from potential users. As a result, it is very difficult for users to figure out any potential accuracy degradation or abnormal behaviors of given ML models. In this work, we propose a new technique named APPLE to explain each ML prediction at the resolution level of circuit elements. To the best of our knowledge, this is the first effort to explain ML predictions on circuit layouts. It provides a significantly more reasonable, useful, and efficient explanation for lithography hotspot prediction, compared with the highest-cited prior solution for natural images.**

I. Introduction

In recent years, many machine learning (ML) techniques have been proposed in the VLSI design flow, ranging from logic synthesis [1] to physical design [2] and design for manufacturability (DFM) [3]. Take DFM as an example, as transistor feature sizes shrink, the manufacturing yield is increasingly affected by the variation in the lithographic process. As a result, manufacturing defects (i.e. lithography hotspots) may arise for some sensitive layout patterns, and thus detection of such defects is important before tape-out. The conventional lithography hotspot detection with lithographic simulation is accurate but very slow. Recent studies on ML solutions show that convolutional neural networks (CNN)-based lithography hotspot detectors can achieve orders-of-magnitude speedups while maintaining a reasonably high accuracy [4]–[6].

While hundreds of publications [3] have been devoted to developing more advanced or customized deep learning methods in electronics design automation (EDA), however, one essential question from potential model users seems to be neglected by many practitioners: *Why shall I trust your ML prediction on my circuit*? This question roots in the difficulty in explaining ML solutions, especially complex deep learning models like CNNs. Currently, model users are only provided with predictions of their circuit quality/problem without any extra information. It is very difficult for users to figure out any potential accuracy degradation or abnormal behaviors of given ML models. As we will discuss, developers often overestimate their models, and unexpected accuracy degradation is not rare in practice. Due to such a lack of explanation on predictions, users inevitably have much less trust in ML predictions compared with traditional simulation tools, especially when wrong predictions can directly affect key design objectives like the manufacturing yield. We believe this neglected problem will become a major obstacle that prevents the wide adoption of ML techniques in the EDA industry.

To the best of our knowledge, this work APPLE is the first attempt to explain ML predictions on circuit layouts. It is a model-agnostic solution that can be directly built on top of any existing ML solutions. In this paper, we primarily target lithography hotspot detection [4]–[6], a representative ML for EDA application on layout, which has received wide attention. Explaining ML predictions with APPLE brings at least three benefits beyond what is possible in prior arts, as we discuss below.

First, explaining predictions allows users to inspect any prediction and choose whether to trust it during inference. In practice, unexpected accuracy degradation may arise when ML models are applied to essentially new test circuits or patterns never seen by the model during training [7], [8]. It is a fallacy to believe ML models will always provide the same level of accuracy demonstrated in validation when it is eventually deployed "in the wild" for real applications. In many ML for EDA works, the held-out validation/test dataset for accuracy evaluation may be an insufficient representation of real model inputs. Such validation data is often too similar to the training data, since they are usually collected in a similar way, from similar designs, and using the same technology node and design flow. But currently there are no techniques that support developers/users to know for which type of new test circuits, their ML model is no longer applicable. APPLE mitigates this challenge by enabling users to inspect a prediction before trusting it.

Second, explaining predictions enables robust ML models against potential malicious attacks. In recent years, malicious attacks including adversarial [9] and backdoor [10] attacks prove effective on ML models for layout. They are especially threatening for models on lithography since it is the last step before tape-out. A latest work [11] proposes robust models against adversarial attacks on lithography, but the backdoor attack is still unsolved. With APPLE, abnormal model behaviors caused by backdoor attacks can be detected.

Third, besides benefiting model users at the inference stage, explaining predictions also provides extra guidance to the development of ML models. Besides evaluation with validation accuracy, developers can inspect whether an ML model is trained to make predictions based on the pattern of interest (i.e. hotspot patterns), instead of other selection bias[1] in samples.

To avoid confusion, we hope to emphasize that there is an essential difference between APPLE and potential objective-detection ML models that directly predict hotspot locations. Instead of being a new ML model, APPLE is a lightweight model-agnostic explainer that can be directly applied on top of any existing ML models, which may perform simple classifications. As an analytical method requiring no training itself, APPLE leaves maximum flexibility to ML model developers or users, who can decide which types of ML to build or use, entirely based on their own demand and available resources.

Our main contributions are summarized as follows:

- We propose a new technique named APPLE[2] to explain the ML prediction at the resolution level of circuit elements. To the best of our knowledge, this is the first work devoted to explaining individual ML prediction on circuit layouts.
- APPLE can explain each individual ML prediction by annotating the ML model's *focus region*, which means a small portion of input sample that has the largest impact on each ML prediction.
- APPLE provides a reasonable, useful, and efficient explanation of ML prediction. Compared with the most widely adopted solution for natural images, it annotates hotspot patterns multiple times more accurately and runs 30× faster. Its explanation also enables the inspection of unexpected accuracy degradation and malicious attacks. This is beyond the capability of prior works.
- APPLE is a general solution that can be built on top of any existing ML models. It does not require any knowledge about the target ML model. Therefore, it is maximally compatible with all existing innovations in model structure and feature engineering.

*Corresponding Author

[1]Consider an example of classifying cows and camels. Due to the *selection bias*, pictures of cows are taken in *green pastures*, while camels are in *deserts*.
[2]APPLE is released in: https://github.com/hkust-zhiyao/apple.

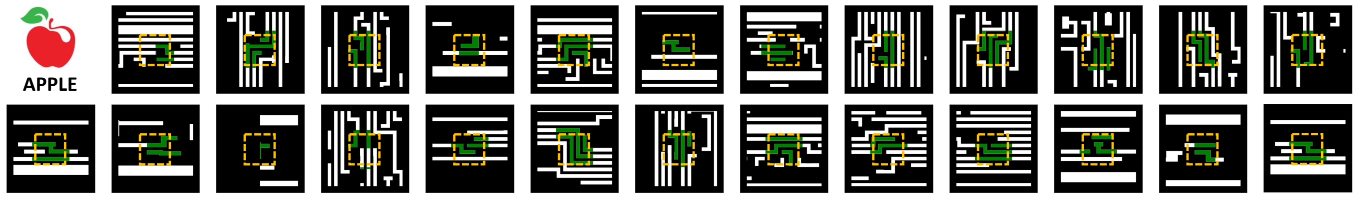


Fig. 1: APPLE-generated explanations on positive predictions from an accurate ML model. The generated model focus regions are annotated in green. All ground-truth hotspot regions are already centered and marked with yellow boxes. This hotspot location is confidential to all algorithms. According to these explanations, this ML model's focus region overlaps surprisingly well with actual hotspots. It means the model's predictions are based on correct hotspot patterns. In practice, selected explanations can be evaluated by comparison with ground-truth hotspot locations from simulation, or by engineers' inspection. Then model users can choose to trust such test predictions.

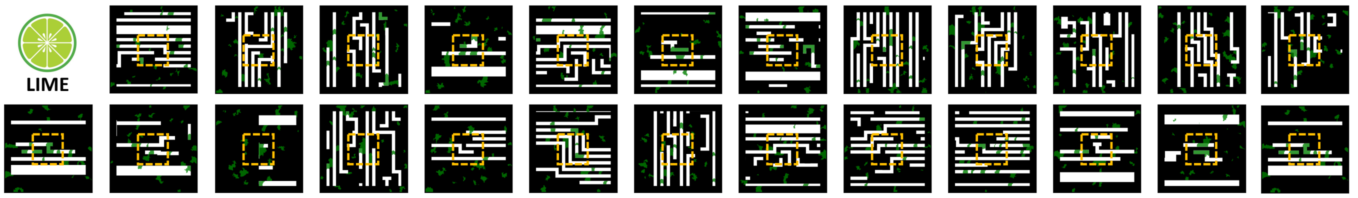


Fig. 2: LIME [12]-generated explanations as a baseline. The ML model, predictions, samples, and the meaning of colors are all the same as Fig. 1. According to LIME's explanation, the model's focus regions fail to indicate reasonable hotspot patterns. It implies the model may not be trustworthy.

II. Preliminaries

A. Machine Learning for Hotspot Prediction

Most ML-based layout hotspot prediction tasks, including the prediction of lithography hotspots [4]–[6], IR drop hotspots, and DRC violation hotspots [13] all share a similar problem formulation. For a circuit layout, task-related features X are extracted from it. When c features are extracted, $X \in \mathbb{R}^{w\times h\times c}$, with each feature in $\mathbb{R}^{w\times h}$. The ground-truth label y is generated with EDA tools. It can be a scalar $\mathbb{R}$ indicating the existence/number of hotspots. Then an ML model F is trained with features X and simulated labels y.

$$F(X) : X \in \mathbb{R}^{w\times h\times c} \to y \in \mathbb{R}$$

Specifically, for lithography hotspot detection, the feature X usually only includes one binary feature directly describing the existence of circuit elements as blockages in a layout clip. The label y is also binary, indicating whether a hotspot exists in this clip. The above problem formulation can be simplified as below.

$$F(X) : X \in \{0,1\}^{w\times h} \to y \in \{0,1\}$$

This formulation is similar to image classification in computer vision. Similarly, most state-of-the-art ML solutions on layouts adopt deep neural networks like CNNs as the model F. The inherent complexity of these deep learning models exaggerates the difficulty in understanding the predictions on circuit layouts.

B. Explainability or Interpretability of ML Models

Although rarely studied in the application on circuits or EDA, the explainability or interpretability of ML models is not an unexplored topic in general ML applications on image and tabular datasets [14]. Existing ML explainers can be categorized based on various metrics, including whether they can explain each individual prediction, whether they are model-agnostic, whether they require white-box access to the ML model, and what data types they handle.

In this work, we select the most widely-adopted[3] ML explainer named LIME [12] as the baseline method to compare with APPLE[4]. For a given image X_{img} and an ML model, LIME highlights input regions that contribute the most to each prediction. Given the input X_{img}, LIME first breaks it down into many contiguous patches of similar pixels (super-pixel). Then it randomly perturbs some super-pixels in inputs by setting them to zero, and generates ML predictions on all these new perturbed samples. Finally, it fits a linear model to evaluate different super-pixels' impact on prediction results, and highlight the most important super-pixel regions.

However, for two-dimensional inputs, existing ML explainers like LIME are designed to handle natural images, while not customized for circuits. In this work, we treat circuit layouts as a special data type and customize APPLE for their unique patterns in multiple aspects. First, APPLE manipulates the input samples by changing circuit elements (i.e. polygons) instead of pixels. This makes the explanation much more reasonable and efficient. Second, APPLE clearly differentiates the selected focus region and the remaining part. Measuring both parts produces more information during explanation and enables the detection of malicious attacks. Third, APPLE tries to minimize the selected region. This customization captures the local pattern of lithography hotspots, and thus contributes to more reasonable explanations. Fourth, APPLE is designed to primarily search the patterns that are more likely to be hotspots. It contributes to APPLE's 30× efficiency over LIME while does not affect the final results. The Fig. 1 and Fig. 2 compare the explanations from these two methods as a preview, with the detailed analysis given in Subsection IV-B.

III. Methodology

A. Concept of Focus Region in Circuit Layout

We develop APPLE to explain the ML prediction $F(X)$[5] on each layout sample X by indicating where the ML model is focusing in this input sample. Such *focus region* denotes a key part of X that has a larger impact on the ML prediction. For positive predictions, the model focus regions are the part of inputs that contributes maximally to the positive prediction, and vice versa. We use the positive prediction $F(X) > 0$ as an example to introduce the algorithm.

For an input sample $X \in \{0,1\}^{w\times h}$, we can extract a local region x from it, with the remaining parts denoted as $X - x$. Both x and $X - x$ are also binary matrices: $x \in \{0,1\}^{w\times h}$, $(X - x) \in \{0,1\}^{w\times h}$. To improve readability, we also describe this extracted region x with a less rigorous notation: $x \in X$.

After the extraction, we can generate model predictions on the selected part as $F(x)$ and the remaining part as $F(X - x)$. The target is to extract the *focus region* $x^* \in X$ that has the maximum impact on the ML prediction. We propose two metrics to evaluate

[3]LIME [12] has >10K citations on Google Scholar and stars on GitHub.

[4]To the best of our knowledge, there are no prior ML explainers customized for IC layout, we thus choose this general solution LIME as our baseline.

[5]Please notice that to capture a large range of prediction value, in this paper, $F(X)$ refers to the raw ML prediction (logit) *before* the final sigmoid function. It means the $F(X) \in (-\infty, +\infty)$ instead of $(0, 1)$.

Algorithm 1 Step 1. Element-Level Focus Region x' Generation

Input: Each test input sample $X \in \{0,1\}^{w \times h}$. A trained ML model F. The ML prediction on this sample $F(X) \in \{0,1\}$.

```
1:  function EVALUATET($x, X, A(x)$)
2:      if $T_{max} < T(x, X)$ in Equation 2 then
3:          $x' = x$, $T_{max} = T(x, X)$
4:  end function
5:
6:  Initialize global variables $x' =$ None, $T_{max} = 0$
7:  Identify all $n$ circuit elements $\{e_1, ...e_n\}$ in sample $X$
8:  for $e_i \in \{e_1, ..., e_n\}$ do
9:      // Iterate $n$ individual element as $x$
10:     Set $x = e_i$, $A(x) = 1^2$, EVALUATET($x, X, A(x)$)
11: for $\{e_i, e_j\} \subset \{e_1, ..., e_n\}$ do
12:     // Iterate $n$ pairs of neighboring elements as $x$
13:     Set $x = e_i + e_j$, $A(x) = 2^2$, EVALUATET($x, X, A(x)$)
14: for $\{e_i, e_j, e_k\} \subset \{e_1, ..., e_n\}$ do
15:     // Iterate $n$ triple-neighboring-elements as $x$
16:     Set $x = e_i + e_j + e_k$, $A(x) = 3^2$, EVALUATET($x, X, A(x)$)
```

Output: The circuit-element-level focus region x' (Notice that this x' is not yet the final solution x^*.)

such impact. First, when the part x is presented to the ML model alone, the prediction $F(x)$ should remain a large positive number. Second, after removing x, the prediction on the remaining part $X - x$ should drop significantly, perhaps to a negative number. It means the target focus region x^* should maximize $F(x)$ and minimize $F(X - x)$. We thus define a new impact function as $I(x) = F(x) - F(X - x)$. Then focus region x^* of each positive ML prediction $F(X) > 0$ is formulated as below:

$$x^* = \text{argmax}_{x \in X} I(x) = \text{argmax}_{x \in X} F(x) - F(X - x) \quad (1)$$

However, Equation 1 is not sufficient. Besides maximizing $I(x)$, a smaller focus region x is also preferable. Without this requirement, the algorithm can directly extract the whole sample with $x^* = X$, thus $I(x^*) = F(X) - F(0)$. It may maximize the impact function $I(x)$ but produces no useful information. Therefore, we define a new area function $A(x)$ to evaluate the regions occupied by x. The final target is scaled by this area function. This new target $I(x)/A(x)$ can be viewed as the impact per unit area. Now the model focus region x^* of each positive ML prediction $F(X) > 0$ is rewritten as:

$$x^* = \text{argmax}_{x \in X} \frac{I(x)}{A(x)} = \text{argmax}_{x \in X} \frac{F(x) - F(X - x)}{A(x)}$$

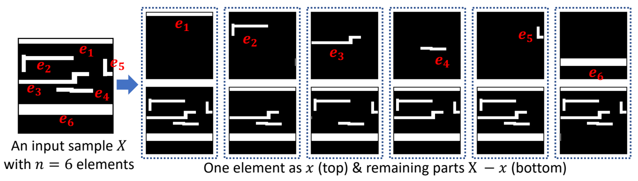
Similarly, for negative prediction $F(X) < 0$, we try to minimize $I(x)$ as a negative number. Therefore we can provide a general form of our target for all predictions, using the absolute value of $I(x)$:

$$x^* = \text{argmax}_{x \in X} \frac{|I(x)|}{A(x)} = \text{argmax}_{x \in X} \frac{|F(x) - F(X - x)|}{A(x)} \quad (2)$$

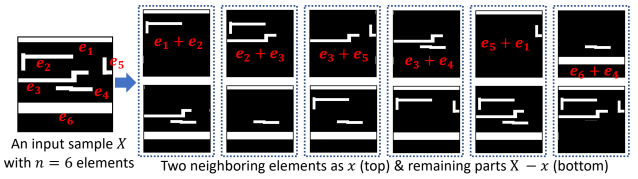
For simplicity, we denote the final target function in the right of Equation 2 as $T(x, X)$. Then $x^* = \text{argmax}_{x \in X} T(x, X)$.

B. Key Circuit Elements as Focus Region - Step 1

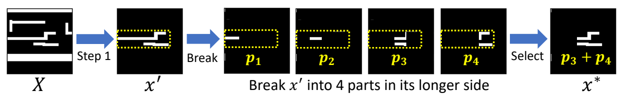
Now we introduce how to implement Equation 2 to explain each ML prediction $F(X)$ with generated model focus region x^*. For better resolution and efficiency of the explainer, the implementation is decomposed into two similar steps. In the first and most important step, the extracted part x is restricted to be *complete* circuit elements (i.e. polygons) from X. The intermediate selected focus region that maximizes $T(x, X)$ in this step is denoted as x'. In the second step, which is covered in next Subsection, the selected x' will be further broken down into multiple smaller parts. The parts that maximize $T(x, X)$ will be the final focus region x^*.


e_1
e_2
e_3
e_4
e_5
e_6
An input sample X with n = 6 elements
One element as x (top) & remaining parts X − x (bottom)

(a) Examples when x (top) includes one circuit element.


e_1
e_2
e_3
e_4
e_5
e_6
An input sample X with n = 6 elements
$e_1 + e_2$
$e_2 + e_3$
$e_3 + e_5$
$e_3 + e_4$
$e_5 + e_1$
$e_6 + e_4$
Two neighboring elements as x (top) & remaining parts X − x (bottom)

(b) Examples when x (top) includes two neighboring elements.



(c) Selection of final focus region x^* in step 2.

Fig. 3: (a) & (b) Decomposition of a sample X into x and $X - x$ at the circuit-element level in step 1. (c) Selection of focus region x^* in step 2.

This first step is introduced in Algorithm 1. For each sample $X \in \{0,1\}^{w \times h}$, we identify all n circuit elements $e_1, ..., e_n \in \{0,1\}^{w \times h}$ in this sample. Each circuit element is one polygon in X. Therefore, $X = \sum_{i=1}^{n} e_i$ with element-wise matrix addition.

Fig. 3 visualizes the decomposition of an input sample X at the resolution of circuit elements. The algorithm starts with setting x to include different circuit elements. It first iterates each individual circuit element as x, as shown in the Fig. 3(a) and line 8-10 of Algorithm 1. The target is to find the region x that maximizes target function $T(x, X)$ in Equation 2. After iterating through n individual elements, we start to search n pairs of neighboring elements as x, as shown in the Fig. 3(b) and line 11-13 of Algorithm 1. After that, we further search three neighboring elements.

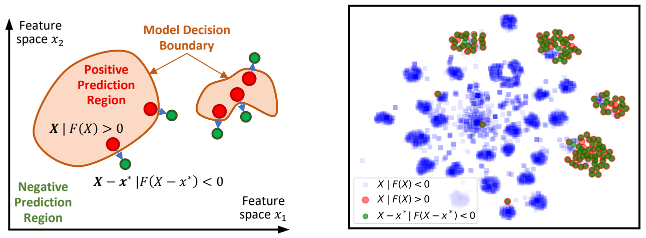
This algorithm incorporates background knowledge about lithography hotspot detection. It only searches neighboring elements for each iteration, since most lithography hotspots are caused by local patterns. Our experiments also validate that using non-neighboring elements as x cannot maximize the target function $T(x, X)$. For similar reasons, it does not search more than three elements in x. It is uncommon for a hotspot to be caused by four elements. Also, we verify that no four elements as x can maximize $T(x, X)$ in the experiment.

As for the area function $A(x)$, as Algorithm 1 shows, for simplicity, we directly set $A(x)$ according to the number of circuit elements in x. To better encourage a smaller x region, we adopt a quadratic function for it. It means $A(x) = 1^2$ if x includes one element, $A(x) = 2^2$ for two elements, $A(x) = 3^2$ for three elements, etc.

In this algorithm, most computation cost comes from the repetitive inference with ML models. For each sample with n circuit elements, since we select only neighboring elements, the number of times it invokes the ML model is only in $\mathcal{O}(n)$. More importantly, the n is relatively small (< 20) for lithography benchmarks and we can perform all $\mathcal{O}(n)$ times of ML inference in parallel with one GPU. The parallelized ML inference with GPU is very fast and this is one major advantage of ML for EDA solutions. As a result, APPLE is already a very efficient explainer algorithm and we do not further accelerate it. In the future, if we need a faster APPLE for different tasks on larger input samples, it is possible to start with an even coarser-grained selection to gradually narrow down the scope of x. Another potential faster solution is to randomly select x instead of iterating through all circuit elements.

C. Key Circuit Parts as Focus Region - Step 2

In the first step, we have selected x', which includes individual or several neighboring circuit elements that maximize $T(x', X)$. We are



(a) (b)

Fig. 4: Distribution of positive sample X and remaining parts $X - x^*$ that are predicted negative. (a) The conceptual visualization of X and $X - x^*$. (b) Visualization of real samples [15] with t-SNE [16]. Positive samples X and corresponding $X - x^*$ overlap well in the distribution.

still not satisfied with such resolution. In this subsection, we propose to further break down the selected circuit elements x', then only select its key parts that have the highest impact on ML prediction.

The selected x' is broken down into four equal-size parts along its longer side. This process is visualized in Fig. 3(c). Then very similar to finding elements x' from X in Algorithm 1, this step further finds the parts x^* from x' that maximizes $T(x, X)$. For this second step, similarly, $A(x) = 1^2$ for one part, $A(x) = 2^2$ for two parts, etc. The starting point x' consists of all four parts in this second step. This step requires even less computation and is faster than step 1. After selection, the key parts x^* is the final output of APPLE, indicating the focus region of each ML prediction $F(X)$.

D. Understanding the Method from Another Perspective

In this work, we view the output of APPLE as the ML model's focus regions that have large impact on each prediction. Besides this, we can also understand our method from another perspective, which is about perturbations across ML model's decision boundary.

For a positive prediction $F(X) > 0$, the goal includes minimizing $F(X - x^*)$. Then this focus region x^* can also be viewed as a tiny perturbation on X towards the ML model's decision boundary, targeting a most negative prediction on $X - x^*$. As we will show in results, for the majority of positive predictions, the corresponding $F(X - x^*) < 0$. This perspective is illustrated in Fig. 4(a), with real measurement on benchmark [15] shown in Fig. 4(b). Each positive test samples X (in pink) and its corresponding $X - x^*$ (in green) are very similar and overlap with each other in the data distribution. This tiny perturbation x^* successfully flip model's positive prediction on X to negative on $X - X^*$. This concept is a bit similar to malicious attacks like backdoor and adversarial attacks. This perspective helps understand why this method can be applied to detect attacks.

IV. Experimental Results

A. Experiment Setup

In the experiment, we adopt representative benchmarks [10], [15] for lithography hotspot detection to evaluate our algorithm. They are summarized in Table I and named B1, B2, and B3 for simplicity. We first adopt the mostly widely-adopted ICCAD'12 benchmark B1 [15] to validate APPLE and compare with baseline method LIME[6]. This benchmark has been adopted in many previous works on lithography detection [4]–[6], [17].

To thoroughly evaluate our methods, we further build more complex and challenging samples based on B1 to generate a new benchmark named B2. Details about benchmark B2 will be introduced in Subsection IV-D. Finally, we will use B3 [10] to study the application of APPLE for backdoor attacks. The B3 benchmark [10] is exactly the dataset used in prior backdoor attack studies [10]. In Table I, the HS means the number of positive samples with hotspots and NHS means the number of non-hotspot negative samples.

[6]B1 includes five sub-benchmarks. When measuring a metric on B1, like prior works, we first evaluate it on each sub-benchmark, then average five values. Also, the number of samples in B1 may slightly differ in different works.

TABLE I: Statistics of Three Lithography Datasets

Benchmarks	Training		Testing	
	HS	NHS	HS	NHS
B1. ICCAD'12 [15]	1303	17441	2750	14458
B2. Complex benchmark [15]	4167	13893	1286	14614
B3. Backdoor attack [10]	787	19213	820	19180

In the experiment, the APPLE method is implemented with Python3. All CNN-based ML models are implemented with PyTorch 1.12 [18]. The LIME [12] baseline is directly adopted from its open-sourced repo on Github. The experiment platform uses Intel Xeon Platinum 8375C CPU and one Nvidia GeForce RTX 3090 GPU for both APPLE and LIME. To balance the accuracy and speed of LIME, the number of new samples to generate and infer for each prediction, as a hyper-parameter, is set to 300 in the LIME baseline.

For benchmark B1, the ground-truth hotspot location of every positive sample X is already centered [15]. Therefore, it provides us with an excellent ground truth to evaluate whether an ML model is focusing on the correct pattern. This information of hotspot being in the center is strictly confidential during the development of ML models and deployment of the APPLE method.

B. Visualization of Different Explainers

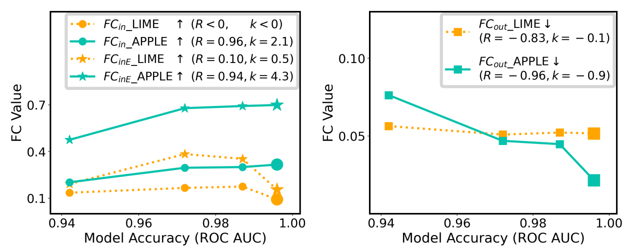
We first train a very accurate CNN-based model on benchmark B1 and achieve the test accuracy with ROC AUC $> 99\%$. Its true positive rate, also named accuracy, equals 98% when the false positive/alarm rate $< 5\%$. This can be regarded as very accurate according to existing studies [4]. We then apply APPLE on this accurate model's positive predictions on B1 test dataset, with results already shown in Fig. 1. We first focus on explaining positive predictions since they have clear ground-truth hotspot locations and thus allow our straightforward inspections. In Fig. 1, the model-focus region x^* generated by APPLE is annotated in green color. The already centered ground-truth hotspot locations are indicated by a yellow squared box. We emphasize again that the information of hotspot being in the center is confidential. According to Fig. 1, the model focus regions (in green) overlap surprisingly well with the real hotspots (in yellow box). The result indicates that: 1) According to APPLE, this accurate ML model is making predictions based on correct hotspot patterns. By evaluating selected focus regions through engineers' inspections or simulations, model users are encouraged to trust the prediction. 2) Seems APPLE is able to generate a very reasonable explanation of ML model's focus regions. We will further validate this.

In comparison, Fig. 2 visualizes the explanation from the widely-adopted baseline LIME [12] on exactly the same ML model's predictions. Although some overlap still exists between LIME-generated focus regions and actual hotspot regions, there are many other regions that are obviously irrelevant to the actual hotspots. It indicates that according to LIME, the accurate ML model is not focusing on correct hotspot patterns. Different from APPLE's explanation in Fig. 1, it may imply the prediction is not trustworthy.

In fact, it is difficult to give a definite and standard answer on which explanation of ML prediction is correct. But we can still make evaluations based on which explanation is more useful and reasonable. First, APPLE indicates that ML predictions are trustworthy. It is confirmed by the high test accuracy. In real application scenarios where test accuracy is unknown, APPLE's conclusion provides correct guidance to model users. In comparison, LIME's explanations in Fig. 2 could be misleading. Second, from the perspective of human designers, the regions annotated by LIME are messy and fail to indicate actual lithography unfriendly patterns. For the ML model with a high test accuracy, such an explanation is rather unreasonable.

C. Quantitative Measurement of Different Explainers

Besides the qualitative analysis of visualized patterns, we further propose a series of new metrics named FC (focus) to quantitatively measure whether an ML model's positive prediction is focusing on correct hotspot patterns in input X, according to a given explainer.



(a) (b)

Fig. 5: FC metrics vs test accuracy for APPLE and LIME. Measured on multiple ML models. APPLE shows a strong correlation between accuracy and FC metric (trustworthy score from explainer). Also, APPLE's FC metrics on the most accurate model are much better than LIME's.

This FC metric can also be viewed as a score about whether a prediction is trustworthy, according to an explainer.

$$FC_{in} = \frac{\text{Hotspot Region } \cap \text{ Focus Region } x^*}{\text{Hotspot Region}} \uparrow$$

$$FC_{inE} = \frac{\text{Elements in Hotspot Region } \cap \text{ Focus Region } x^*}{\text{Elements in Hotspot Region}} \uparrow$$

$$FC_{out} = \frac{\text{Non-Hotspot Region } \cap \text{ Focus Region } x^*}{\text{Non-Hotspot Region}} \downarrow \quad (3)$$

These above FC metrics calculate the amount of overlap between real ground-truth hotspot regions and focus regions x^*. As annotated by arrows (↑↓), higher FC_{in}, FC_{inE} values and lower FC_{out} value indicate that model focus regions overlap better with correct hotspot patterns. Besides the aforementioned accurate model with $AUC > 0.99$, we further train several ML models with lower test accuracies. Then we apply APPLE and LIME to explain all these models predictions on test dataset, with results shown in Fig. 5[7].

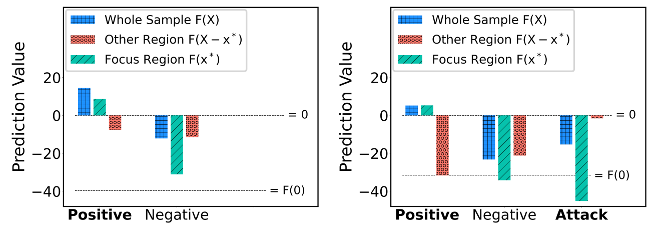
Fig. 5 shows test accuracies of four different ML models versus FC metrics when explaining their predictions using both APPLE and LIME. The most accurate model, i.e. rightmost points in Fig. 5, is what we have analyzed previously. We have two important observations: 1) For the most accurate model, FC_{in} and FC_{inE} of APPLE are significantly higher than LIME, and FC_{out} of APPLE is lower. This is consistent with the observation that APPLE correctly marks the real hotspots for accurate models. 2) For those less accurate models, FC metrics of APPLE in y-axis drop significantly and the trend is consistent with test accuracy in x-axis. This is an excellent and useful correlation. It indicates that APPLE's explanation on each prediction can reflect actual test accuracy. In comparison, LIME's FC metrics remain low and flat for models with different test accuracy.

Fig. 5 further quantitatively measures this relationship between the model test accuracies in x-axis and FC metrics in y-axis. It reports both the correlation R and the slope k for both APPLE and LIME. APPLE shows correlation $R = 0.96$ for FC_{in} and $R = 0.94$ for FC_{inE}. As mentioned, this correlation is much higher than LIME. Such a strong correlation between FC and test accuracy makes APPLE a useful explainer. In real applications, users may inspect explanations provided by APPLE, then decide whether the ML prediction will be accurate. Besides correlation R, the difference in slope k is also obvious. The slope k of LIME in Fig. 5 is very low. It means the actual difference in LIME's FC metrics for different models is very small. Therefore, it is infeasible to rely on LIME to decide whether predictions are trustworthy.

[7]For a fair comparison, we perform measurement on the same set of test samples for different models (points) in Fig. 5. To ensure this, we measure the test samples that are predicted as positive by all models.

TABLE II: Runtime Comparison (both with GPUs)

	APPLE	LIME
Runtime to explain each prediction	0.53 seconds	18.2 seconds



(a) Benchmark B1 [15] (b) Benchmark B3 [10]

Fig. 6: The prediction value averaged over all positive and negative predictions from (a) benchmarks B1 and (b) benchmark B3 for backdoor attack. It includes the prediction of whole input sample $F(X)$, focus regions $F(x^*)$, and other parts $F(X - x^*)$. The focus region x^* shows a high impact in both normal positive prediction and wrong negative prediction on attack samples.

We have more interesting observations in Fig. 6(a), which measures both positive and negative predictions. It averages the accurate ML model's predictions on different regions, including the whole test sample $F(X)$, the focus region $F(x^*)$, and remaining parts $F(X - x^*)$. For positive prediction, as Fig. 6(a) shows, focus region x^* alone receives a similar level of positive prediction to X, while remaining part $X - x^*$ alone is predicted negative. It verifies the impact of selected x^* on the positive prediction. As for negative predictions, in comparison, their focus regions x^* have much less impact. After removing x^*, the prediction on remaining part $X - x^*$ is still close to the total prediction on X. In this negative case, we cannot even say model is 'focusing' only on this part, since remaining parts alone receive a similar prediction. This result means: 1) For positive predictions, local patterns like x^* can determine the existence of hotpots. 2) For negative predictions, the non-hotspot prediction cannot be made only based on local patterns. This difference is intuitive and consistent with the process of identifying hotspots. In Subsection IV-E, we will further introduce a different pattern of negative predictions with Fig. 6(b), where local region indeed has high impact, but that is actually a sign of malicious attack.

Besides providing reasonable and useful explanation, explainers should also be efficient. Table II compares the runtime of APPLE and LIME for explaining one prediction $F(X)$. APPLE is $> 30\times$ faster than this baseline method. As mentioned, APPLE performs only $\mathcal{O}(n)$ ML inference all in parallel with one GPU. In comparison, it takes LIME a long time to generate and infer hundreds of new samples by perturbing super-pixels. In practice, a typical application scenario is to randomly sample predictions and examine the explanations on these samples with human engineers. Therefore, we do not expect generating explanation for a huge number of test samples and thus APPLE's efficiency should be sufficient.

In addition, we demonstrate that APPLE is a general solution that applies to existing innovations in model and feature engineering. We apply APPLE to a different ML solution, which uses a feature preprocessing technique named discrete cosine transform (DCT) [4]. The APPLE-generated explanations show a same level of FC values as Fig. 5. This DCT-processing plus CNN model gets $FC_{inE} = 0.64$ with $AUC > 99\%$. By running DCT and model together with GPU, the efficiency of APPLE also remains largely the same.

D. More Challenging Examples

Despite the superior performance of APPLE in benchmark B1, some researchers may suspect that: *1) The samples from B1 are rather simple with a small number of circuit elements; 2) All ground-truth regions are in the center. Although this is confidential, such patterns may still somehow benefit APPLE.* To mitigate such concerns and fully evaluate the capability of APPLE, we construct a more complex and challenging dataset named B2, by combining multiple samples from B1. The area of each new complex sample in B2 is 4 times as large as B1, containing 1 to 4 randomly selected samples from B1. The label of the new sample is positive if it includes a positive clip.

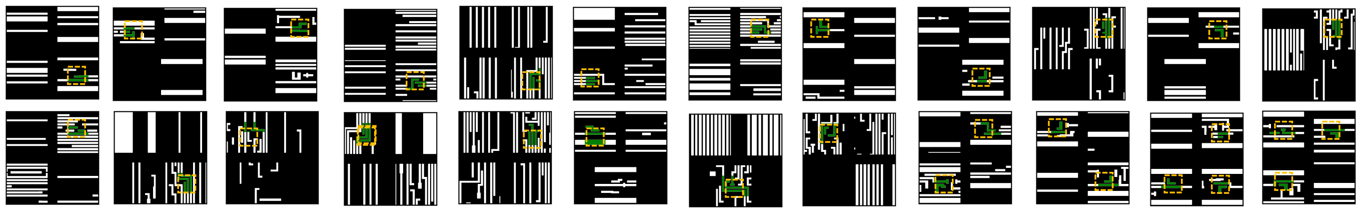


Fig. 7: APPLE-generated focus regions (green) for B2 benchmark. They still overlap very well with ground-truth hotspots in these complex samples.

In this way, there are more circuit elements $\{e_1, ..., e_n\}$ in each new sample, and the ground-truth regions will be in different locations. A new ML model is trained on this more complex dataset to achieve high test accuracy AUC $> 99\%$. Fig. 7 visualizes APPLE's outputs on the B2. Although ground-truth hotspot regions are no longer in the center and samples get more complex, APPLE-generated focus regions still overlap very well with real hotspot patterns.

As Fig. 7 shows, to make the task more challenging, we generate some samples with multiple hotspots, although this may be rare in practical applications. APPLE can also well handle this after we slightly revised the explanation flow. Specifically, after APPLE's explanation of a positive prediction, we remove the APPLE-indicated focus regions x^* from the input X, then make ML prediction again on this new input $X - x^*$. If the new prediction is still positive, then it indicates that there is one more hotspot in $X - x^*$, thus APPLE is applied again. Such iteration continues until the prediction turns negative. As Fig. 7 shows, APPLE can correctly identify multiple hotspots in the same sample as focus regions.

E. Prevention of Malicious Attacks

Besides deciding whether normal predictions are trustworthy when test labels are unknown in real applications, another important application of APPLE is to prevent malicious attacks on ML solutions in circuit design by detecting abnormal behaviors of ML models.

In backdoor attack [10], malicious attackers secretly poison training data by adding extra patterns/triggers in some negative training samples. After training with poisoned data, ML models predict all samples with backdoor triggers as negative. As a result, the model fails to detect hotspots in positive samples when they include triggers. To study this, we first train ML model with poisoned training data from B3 for backdoor attack [10]. The model is accurate on normal test samples, but make mistakes on actually-positive test samples with backdoor triggers, wrongly predicting them as negative. The backdoor triggers in benchmark B3 are all in the shape of ✝.

We apply APPLE to explain these wrong negative predictions caused by backdoor triggers, as shown in Fig. 8. Now the focus regions x^* are those contribute the most to the negative prediction. We annotate the negative focus regions in red. It turns out APPLE-generated focus regions in Fig. 8 overlap perfectly well with the actual backdoor triggers in the ✝ shape. In other words, APPLE reveals that it is the trigger pattern that contributes the most to all these wrong negative predictions. By inspecting explanations from APPLE, experienced engineers can identify backdoor triggers in test samples. Also, when many focus regions of negative predictions show exactly the same pattern, they may be triggers for backdoor attacks.

Besides identifying backdoor triggers by their similarity or manual inspection, APPLE supports more efficient and automated identification of these backdoor attacks. Fig. 6(b) reports the prediction values of different regions for three different types of predictions: positive predictions, negative predictions, and wrong negative predictions on backdoor-attack samples. The overall predictions in Fig. 6(b) is more negative than Fig. 6(a) since B3's training data is more biased than

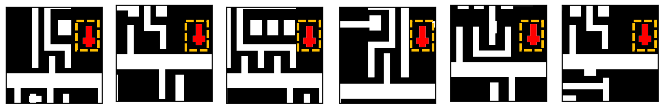


Fig. 8: APPLE-generated model-focus regions (red) on samples with backdoor triggers from B2 backdoor benchmark. These samples are with positive labels while predicted to be negative by poisoned ML models.

B1. As mentioned, the APPLE's focus regions x^* on backdoor attack samples are exactly the triggers. As Fig. 6(b) shows, the model prediction on such focus region $F(x^*)$ in attack samples is significantly smaller than other negative predictions, or even smaller than $F(0)$, the prediction of hotspot on a completely empty input. Also, the prediction on remaining parts $F(X - x^*)$ of attack samples is close to zero. This reveals the poisoned ML model's strong and abnormal tendency to classify the trigger x^* as negative, while being much less affected by other regions of attack samples. It provides us with an excellent sign to identify malicious attacks.

V. Conclusion

In this paper, we present APPLE to explain ML predictions on circuit layouts. We focus primarily on lithography hotspot detection. The algorithm is well customized for the data pattern in this task, and it provides much better explanations over existing solutions. It is also maximally compatible with existing ML solutions. In the future, we expect more explainers on other tasks and beyond circuit layout. The improvement in interpretability will certainly boost users' trust, and thus pave the way to a wider adoption of ML for EDA techniques.

Acknowledgement

This work is partially supported by the Hong Kong Research Grants Council (RGC) ECS Grant 26208723, National Natural Science Foundation of China (62304192, 92364102), and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

References

[1] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *DAC*, 2018.
[2] A. B. Kahng, "Machine learning applications in physical design: Recent results and directions," in *ISPD*, 2018.
[3] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "MLCAD: A survey of research in machine learning for CAD keynote paper," *TCAD*, 2021.
[4] H. Yang, J. Su, Y. Zou, Y. Ma *et al.*, "Layout hotspot detection with feature tensor generation and deep biased learning," *TCAD*, 2018.
[5] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: a deep learning approach," *JM3*, 2017.
[6] H. Yang, W. Zhong, Y. Ma, H. Geng *et al.*, "VLSI mask optimization: From shallow to deep learning," *Integration*, 2021.
[7] Z. Xie, J. Pan, C.-C. Chang, J. Hu, and Y. Chen, "The dark side: Security concerns in machine learning for EDA," *TCAD*, 2022.
[8] G. R. Reddy *et al.*, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *ICCAD*, 2019.
[9] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Young, R. Karri, and S. Garg, "Adversarial perturbation attacks on ML-based CAD: A case study on CNN-based lithographic hotspot detection," *TODAES*, 2020.
[10] K. Liu, B. Tan *et al.*, "Poisoning the (data) well in ML-based CAD: A case study of hiding lithographic hotspots," in *DATE*, 2020.
[11] J. Pan, C.-C. Chang, Z. Xie, J. Hu, and Y. Chen, "Robustify ML-based lithography hotspot detectors," in *ICCAD*, 2022.
[12] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?" Explaining the predictions of any classifier," in *KDD*, 2016.
[13] Z. Xie, Y.-H. Huang *et al.*, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *ICCAD*, 2018.
[14] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable AI: A review of machine learning interpretability methods," *Entropy*, 2020.
[15] J. A. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *ICCAD*, 2012.
[16] L. Van der Maaten *et al.*, "Visualizing data using t-SNE." *JMLR*, 2008.
[17] G. R. Reddy *et al.*, "Enhanced hotspot detection through synthetic pattern generation and design of experiments," in *VTS*, 2018.
[18] A. Paszke, S. Gross, F. Massa, A. Lerer *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *NeurIPs*, 2019.