Deep Learning-Driven Simultaneous Layout Decomposition and Mask Optimization

Wei Zhong, Shuxiang Hu, Yuzhe Ma, Member, IEEE, Haoyu Yang, Xiuyuan Ma, and Bei Yu, Member, IEEE

Abstract—Combining multiple patterning lithography (MPL) and optical proximity correction (OPC) pushes the limit of 193-nm wavelength lithography to go further. Considering that layout decomposition may generate plenty of solutions with diverse printabilities, relying on conventional mask optimization (MO) process to select the best candidate for manufacturing is computationally expensive. Therefore, an accurate and efficient printability estimation is crucial and can significantly accelerate the layout decomposition and MO (LDMO) flow. In this article, we propose a convolutional neural network (CNN)-based prediction and integrate it into our new high-performance LDMO framework. The optimization process can be considerably improved as the decomposition quality has been inferred in the early phase. To facilitate the network training and ensure better estimation accuracy, we develop sampling strategies for both layout and decomposition. Moreover, we enhance the layout sampling approach by adopting autoencoder to distance evaluation that promises superior sampling results. The experimental results demonstrate the effectiveness and the efficiency of the proposed algorithms.

Index Terms—Convolutional neural network, Design for manufacturing, layout decomposition, mask optimization.

I. INTRODUCTION

THE SHRINKAGE of device feature size has reached the resolution limit of the 193-nm wavelength lithography, thus various resolution enhancement techniques (RETs) are heavily applied to maintain a good printability when transferring patterns from mask to wafer, among which multiple patterning lithography (MPL) and optical proximity correction (OPC) are two very promising approaches.

MPL is currently widely applied to enhance the resolution in the industry. The key step in litho-etch-litho-etch (LELE)-type manufacturing, layout decomposition, mask optimization. (a) Different decomposition optimization results of the same layout. (b) Corresponding decomposition convergence comparison between DS and MO.

MPL is the layout decomposition which assigns the conflicting patterns on a layer to separated masks for manufacturing. To achieve better decomposition quality, various methods have been proposed [1]–[4]. OPC or mask optimization (MO) is able to handle the optical distortions in subwavelength lithography by refining the pattern shapes on a mask. Various inverse lithography technology (ILT)-based approaches are proposed to implement the OPC process [5]–[8].

After decomposing a layout, multiple solutions can be obtained, as shown in Fig. 1 (a). To further enhance the printability, MO is performed. Since the MO is a subsequent step of the layout decomposition, the final quality is determined, to a large extent, by the layout decomposition result. Fig. 1 (b) shows corresponding trajectories during MO of different decomposition results. It is observed that the printability is not consistently good or bad for a given instance. Only after the entire process is completed can we tell the good ones from the bad ones. However, it is computationally expensive to run all solutions through the MO process due to the overhead imposed by lithography modeling. Recently it has been demonstrated that a simultaneous layout decomposition and MO (LDMO) framework can ease the gap and obtain high printability masks in a unified way [9], in which the final masks are generated by the collaboration of MO engine and...
of the training set, the decomposition sampling approach also uses n-wise method to obtain training-friendly decomposition results. The main contributions of this work are as follows.

1) We propose a CNN predictor to estimate the printability before optimizing masks.
2) We combine the no-odd graph and n-wise method to generate layout decomposition candidates more efficiently.
3) We develop a set of sampling approaches to select the representative decomposition as the training set. The comparison with random sampling strategy shows the superiority of our sampling method.
4) We improve the sampling strategy by integrating autoencoder into our training flow to ensure better layout similarity evaluation.
5) Experimental results show that our framework outperforms other previous works and reduces edge placement error (EPE) by 81.9% in comparison with state-of-the-art methods.

The remainder of this article is organized as follows. Section II introduces the background of double patterning lithography and gives the problem definition. Section III describes the optimization framework. Section IV shows the details of training the prediction module, including layout sampling, decomposition sampling, and CNN training steps. Experimental results are detailed in Section V. Sections VI and VII, respectively, present the discussion for future work and conclusion.

II. PRELIMINARIES

Notations and their descriptions in this article are listed in Table I. The task of MO for double patterning lithography is to generate a pair of optimized masks such that the final printed image $\mathcal{T}$ and the target image $\mathcal{T}'$ are as close as possible. The lithography simulation process can be represented by two models: 1) optical model and 2) photo-resist model.

The theoretical basis of optical model is the Hopkins diffraction model [16] which has been widely applied to a partially coherent imaging system. It is given by

$$I_i(x, y) = \sum_{k=1}^{N^2} w_k \cdot |M_i(x, y) \otimes h_k(x, y)|^2$$  \hspace{1cm} (1)$$

where $M_i$ is a given mask to calculate the aerial image $I_i$, $h_k$ is the $k$th optical kernel, $w_k$ is the weight of $h_k$. The system contains $N^2$ illumination sources in total.

Due to the high complexity of the Hopkins diffraction model, a singular value decomposition (SVD) method [17]...
is adopted for approximation, which is described as
\[ I_i(x, y) \approx \sum_{k=1}^{K} w_k \cdot |M_i(x, y) \otimes h_k(x, y)|^2 \]  
where \( K \) is the total kernel number we selected to approximate the optical system.

The photo-resist model controls the printing of the final wafer image. The shape is printed to the final image, when the intensity is greater than a given threshold, as shown in
\[ T_i(x, y) = \begin{cases} 1, & \text{if} \ I_i(x, y) \geq I_{th} \\ 0, & \text{if} \ I_i(x, y) < I_{th}. \end{cases} \]  
In order to use ILT to optimize masks, a new variable \( P_i \) is introduced and the sigmoid function [18] is applied to binary parameter \( P \).

\[ M_i(x, y) = \frac{1}{1 + e^{-\theta_m P_i(x, y)}}. \]  
In this way, the binary mask \( M \) is expressed with unbound parameter \( P \). \( \theta_m \) is the coefficient to control the slope of the sigmoid function. Similarly, the relaxed photo-resist model can be presented as
\[ T_i(x, y) = \frac{1}{1 + e^{-\theta_l (I_i(x, y) - I_{th})}}. \]  
In our implementation, hyperparameters \( \theta_m, \theta_l \) are set to 8, 120 to achieve better optimization performance. \( I_{th} \) is set to 0.039 according to [19]. In double patterning ILT, the printed image is organized in the following form:
\[ T(x, y) = \min[T_1(x, y) + T_2(x, y), 1]. \]  
Then we can derive the gradient of \( T(x, y) \) with respect to \( P_i(x, y) \), and update corresponding \( M_i(x, y) \) by performing ILT to reshape masks and obtain better result \( T \). More details about the gradient formulation can be seen in [9].

**Definition 1 (EPE):** EPE measures the manufacturing distortion by the edge displacement between the printed image and the target layout. A checkpoint will be marked as an EPE violation if its EPE greater than a given threshold.

**Definition 2 (Squared \( L_2 \) Error):** Squared \( L_2 \) error measures the difference between the printed image \( T \) and the target image \( T' \), which is defined as \( \| T - T' \|_2^2 \).

EPE is one of the most important criteria of image printability. ILT process reduces the squared \( L_2 \) error in each iteration to minimize the number of EPE indirectly, and a smaller squared \( L_2 \) error indicates a better layout printability. In our work, both EPE and squared \( L_2 \) error are selected as printability metrics.

The task of layout decomposition for double patterning lithography is to generate a pair of decomposed masks so that the masks follow the design rule. This process can be presented like
\[ f_{\text{decomp}}(T') = [M_1, M_2] \]  
where \( T' \) is the target image and \( M_1, M_2 \) are decomposition result. The subsequent ILT process generates optimized masks to minimize the differences between printed image and target image, (8) describes this process
\[ f_{\text{ILT}}(M_1, M_2) = f_{\text{ILT}}(f_{\text{decomp}}(T')) = T \]  
where \( T \) is the printed image after ILT optimization. We can evaluate its quality by organizing the combination form of EPE and \( L_2 \) Error. Thus, the \( f_{\text{ILT}} \) converts to a continuous function.

In this work, we try to use CNN to regress the continuous function derived from (8) and use the network to supervise the process of (7) to generate a better decomposition. For simplicity and to demonstrate the methodology effectiveness, we use constant photo-resist model instead of variable threshold model. Since self-aligned double patterning (SADP) technology can be applied for metal layer, we only focus on the contact layer.

Based on the above definitions and discussions, the LDMO problem can be described as follows.
Problem 1 (LDMO): Given a target image $T'$, decompose the layout to obtain mask candidates that result in fewer EPE violations upon MO.

III. OPTIMIZATION FRAMEWORK

The overall flow of our optimization framework is shown in Fig. 2. It contains two parts: the left part introduces the optimization flow, while the right part details the predictor training used in the printability prediction module of the left part. In this section, we focus on the framework optimization process (the left part flow). First, decomposition candidates are generated according to the input layout. In order to obtain legal decomposition candidates rapidly, we build several no-odd graphs and apply $n$-wise method to avoid generating violated decomposition candidates. Then all candidates are fed into the printability prediction module. The trained CNN scores each candidate and outputs the best layout decomposition. Next, ILT process is used to optimize masks and outputs the optimized final masks. Besides, print violations are checked during the ILT optimization process to avoid printability estimation errors.

A. Decomposition Generation

The decomposition generation module produces decomposed mask candidates for the next module based on the given layout. Since we are seeking for a pair of masks, $M_1$ and $M_2$, from these candidates to achieve the best printability, the generated candidates should contain these high-quality decomposition results. Enumerating all possible decomposition results can tackle this problem, but the time consumption is expensive even though there is a decomposition quality predictor to help select the best one. So we focus on the most promising results.

In our generating strategy, the key to generating a set of high-quality decomposed masks in the layout decomposition phase can be viewed from two different scales. In the macro view, we should solve the conflicts in accordance with the coloring rules. It ensures our decomposition results follow the design rules and discards low-quality results according to our experience. But from the perspective of micro view, coloring rules are coarse constraints we manually made, thus they can not help more in finding a hidden relationship, i.e., selecting the best decomposition among the legal decomposition candidates. Thankfully, due to the powerful modeling ability of deep networks, they are efficient in distinguishing a better result among candidates. By combining eligible decomposition results and through the use of deep networks, we are able to improve the decomposition quality further. This section mainly introduces our generating strategy, as described in Algorithm 1.

As shown in Fig. 3, layout patterns are first divided into three sets: 1) separated pattern ($S_P$ set; 2) appended pattern ($A_P$) set; and 3) normal pattern ($N_P$) set. Based on the distance $d$ of the nearest patterns, the belonging of pattern $E$ is determined by

$$
E = \begin{cases} 
S_P, & \text{if } d \leq n_{\text{min}} \\
A_P, & \text{if } n_{\text{min}} < d \leq n_{\text{max}} \\
N_P, & \text{if } n_{\text{max}} < d.
\end{cases}
$$

![Fig. 3. $S_P$ distribution solution. (a) Constructed weighted graph. (b) No-odd graph solution.](image)

A print violation occurs when the distance between two patterns is less than $n_{\text{min}}$, and it disappears as the distance $d$ increases, but there may still exist the interaction between two patterns until the distance reaches $n_{\text{max}}$. Therefore, patterns in $S_P$ always cause print violations, so they ought to be separated from each other. $A_P$ are the kind of patterns that tend to cause printability issues while $N_P$ have minimal or no effect on the performance compared to the other two types. When generating decomposition candidates, we take different strategies according to the pattern type. In our implementation, $n_{\text{min}}$ is set to 80 nm, and $n_{\text{max}}$ is set to 98 nm.

We first solve the print violations in the macro view. Print violations in set $S_P$ are fixed by allocating the shapes within the set to two separate masks, which is solved by constructing a weighted graph and is converted to a coloring problem. Here, we take patterns in $S_P$ as vertices and the distance among them as the weight of edge, thus a weighted graph is built. But the generated weighted graph may not be a colorable graph. Therefore, in order to optimize the printability and make the generated graph colorable, our strategy is to find a min-weight graph without odd cycles for each component, which is described in Algorithm 2. For each component, we are able to obtain a weighted graph as shown in Fig. 3(a). Obviously, component 1 is not a colorable graph, so we take the component as input $W$, and store its edges in $E$. Edges in $E$ are sorted in ascending order so that we can easily access the edge with minimum weight by taking the first element in $E$. No-edge graph $G$ is initialized according to $W$, which has the same vertices number, but no edges are linked. In each iteration, the first edge in $E$ is added to $G$ only if it does not cause
Algorithm 2 Build Min-Weight Graph Without Odd Cycles

Require: A weighted graph $W$

1. Store edges of $W$ in $E$;
2. Sort $E$ in an ascending order;
3. Initialize a graph $G$ without edges;
4. while $E$ is not empty do
5.  
6.  
7.  
8. end while
9. return $G$;

odd cycles. After handling all edges, a min-weight graph, as shown in Fig. 3(b), is prepared for coloring. Then $G$ can be legally colored by assigning different colors to neighbor nodes. Based on the result of no-odd graph, two neighbor vertices can be assigned to different masks to avoid violations. Besides, the relative position relationship of patterns in the same no-odd graph can be inferred, which provides the basis of the following decomposition analysis.

After solving the $S_P$ distribution, we consider the combination of patterns $A_P$ and $N_P$ in micro view. Another requirement is that we want to reduce the number of decomposition candidates as much as possible. So $n$-wise method is applied to generate representative decomposition candidates, meanwhile limiting the size of the candidate set. The $n$-wise test method (also known as combinatorial test method) has been used to test compiler by Mandl [20]. Usually, it is used to analyze the main factors affecting the experiment with the smallest test set. The main idea of $n$-wise method is to obtain the full factor combination of local areas at the price of giving up the global factor combination strength. Here, $n$ represents how many factors we can test according to the generated arrays. For example, if $n$ is 2, we can find the problem caused by the interaction of two factors.

In our decomposition process, covering all the combinations of patterns is prohibitively expensive which is similar to software testing. An example of two-wise (pairwise) arrays with four patterns is shown in

$$\begin{bmatrix}
\text{factor1} & \text{factor2} & \text{factor3} & \text{factor4} \\
\text{instance #1} & 1 & 0 & 0 & 0 \\
\text{instance #2} & 1 & 1 & 1 & 1 \\
\text{instance #3} & 0 & 1 & 0 & 1 \\
\text{instance #4} & 0 & 0 & 1 & 1 \\
\text{instance #5} & 0 & 1 & 1 & 0 \\
\end{bmatrix}$$

In the generated arrays, each row is an instance of decomposition, each column represents a pattern (factor), and the value determines which masks this pattern belongs to. Picking any two columns, the complete combination of them (00, 01, 10, 11) exists, which means two-wise method reduces the strength of factor combination to minimize the generated arrays meanwhile maintaining the complete combination of any two factors. Naturally, if $n$ is set to the number of factors, the test set becomes Cartesian product of all factors whose size is 16. From the example we can find that the combinatorial explosion of patterns can be well handled.

Based on the no-odd graph result, we can obtain the relative position relationship of $S_P$. In order to build the connection between $S_P$ and $A_P$, we randomly pick a pattern from each connected component as a factor and apply three-wise method together with the patterns in $A_P$. Then we apply two-wise method to patterns in $N_P$, thus, two arrays with different combination strengths are created. We can simply combine the arrays to get the duplicated decomposition results (some lines in the arrays actually stand for the same decomposition, we will solve this problem later). Take the layout in Fig. 3(a) as an example, there are six patterns in $S_P$, two patterns in $A_P$, three patterns in $N_P$ and two connected components, and the corresponding no-odd graph result of the components is shown in Fig. 3(b). We randomly select pattern F in component 1 and pattern H in component 2 to apply the 3-wise method together with patterns in $A_P$ (E and G), the generated $Arrs_1$ can be seen in Fig. 4(a). As for patterns in $N_P$ (J, K and L), two-wise method is used to generate $Arrs_2$ [see Fig. 4(b)]. From Fig. 4(a) and (b), we can see that the number of instances does not grow too much with the number of factors.

We call the combination of $Arrs_1$ and $Arrs_2$ is duplicated decomposition results because although $n$-wise method generates the minimal training set of strength $n$, there are still identical decomposition candidates. The output of the decomposition generation module is a grayscale image with different grayscale levels to represent patterns distributed on different masks. Since the masks are unordered, a layout decomposition can be represented by two different images, as shown in Fig. 4(c). Different colors mean the patterns are distributed on different masks. To solve this problem, we number the layout patterns from left to right and from top to bottom. We manually fix the pattern numbered 1 on $M_1$ so that the two masks become ordered. When generating the decomposition candidates, once pattern numbered 1 is distributed on $M_2$, the value of this row will be reversed. After checking and reversing all rows, we merge the same rows to drop the same decomposition. Note that this operation will not destroy the relative position relationship among patterns. So the total decomposition candidate number should be $\text{size}(\text{mergedArrs}_1) \times \text{size}(\text{mergedArrs}_2)$.
B. Mask Pattern Density Balance

Layout density balance for double patterning lithography is also expected to be considered. The decomposition generation strategy in Section III-A simply combines the arrays, but it may generate imbalanced decomposition candidates. In our approach, we also designed an optional stage to balance the decomposition candidates. In mergedArrays and mergedArrays2, 0/1 respectively which mask the pattern is distributed on. To uniformly map patterns onto two masks, the density rule checking is added before combining arrays. The checking process is implemented by counting how many 0/1 numbers in the two instances (a line in the array is called an instance), which are, respectively, from mergedArrays1 and mergedArrays2. Let $P_0$ denote the 0 count and $P_1$ is the 1 count in two instances, the balanced decomposition should obey the rule that $\frac{|P_1 - P_0|}{\max(P_1, P_0)} \leq 40\%$. The decomposition candidates are discarded if the density checking is not passed.

C. Printability Prediction

Traditional two-stage approaches focus on formulating rules to avoid violations. However, more manufacturing friendly decomposition is hardly obtained by designing the rough restrictions, i.e., build conflict graph by spacing rules, which limits the performance of further MO. The CNN can build the mapping relation from input to output, especially in the image processing field, so they are suitable for dealing with DS problems. Another advantage of CNN is that no matter what kind of searching algorithm is selected to find the best decomposition, the computationally expensive lithography simulation process will be the bottlenecks of these algorithms, but CNN replaces simulation by estimation, hence accelerates the DS process.

The printability prediction module in the left part of Fig. 2 evaluates the decomposition printability by giving candidates scores. The well-trained estimation model is obtained from the right part of Fig. 2, which shows a complete training flow and we will introduce the training process in Section IV.

In the printability prediction module, all decomposition candidates are fed to CNN in the form of grayscale images. A lower score indicates a better printability, so in order to find the best printability after the ILT optimization, this module scores each input and output the decomposition of the minimal score. Considering we have the requirement of reselecting decomposition candidates because printability prediction errors may happen, this module will cooperate with the ILT module to avoid outputting the same decomposition result. ILT module optimizes the masks meanwhile detecting print violations, and the printability prediction module gives the no-repeat decomposition. There are two measures to ensure different decomposition results.

1) Each fixed no-odd graph has two possible decomposition results, so random selecting patterns in each no-odd graph and combine the results can generate different decomposition candidates. Naturally, the selection result should vary with the given candidates.

2) The previous illegal outputs are recorded such that the printability prediction module can skip the same decomposition.

D. ILT Optimization

At this point, the best decomposition result has been obtained. This module optimizes decomposed masks meanwhile detecting print violations. Violations indicate the printability estimation error. Once they are detected, we go back to the decomposition generation step to create new decomposition candidates and use CNN to select another decomposition solution, otherwise, we continue to optimize the masks. Considering the print violations may happen at any iteration of ILT, we detect them every three iterations and the violation detection method is from the discrete optimization part of [9].

ILT process calculate the gradient $g$ of object function $\|T - T'\|_2^2$ with respect to $P_1, P_2$ and update parameters in the form of $P_i = P_i - \text{stepSize} \times g$. Then, we can update $M_i, I_i$ and printed image $T$ with new $P_i$ according to (2)–(6). The ILT optimization will early stop when it removes all EPE violations, or it reaches the max iteration threshold. In our implementation, the iteration threshold is set to 30.

IV. TRAINING OF PREDICTION NETWORK

In this section, sampling strategies and training approaches are detailed. As shown in the right part of Fig. 2, sampling strategies can be divided into two stages: 1) layout sampling and 2) decomposition sampling. We first describe the definitions of layout similarity and cluster layouts based on the distance definitions. After sampling layouts from each cluster, we introduce the decomposition sampling strategy and discuss the model structure.

A. Layout Sampling

In order to achieve the purpose of increasing layout diversity in the training set, our strategy is clustering layouts and sampling them from each cluster. The clustering method gathers similar layouts into a group, so the layouts sampled from each cluster are called representative layouts, as they have a large distance from other cluster elements. But before conducting clustering algorithms, the distance between two layouts needs to be clearly defined. Layout feature extraction has been widely investigated in recent studies. Ma [21] reflects layout distance by XOR operation. Wen et al. [22] used density feature vectors to extract pattern shapes. Yao et al. [23] encoded layout into strings to represent pattern topology. Due to the diffraction and interference of light, printed image quality is largely dependent on the nearby patterns. The distance metric should capture common layout features (e.g., recognize representative pattern distribution, ignore slight layout movement and rotation) and express light propagation property. But previous methods are not a perfect fit for our requirements.

In this section, the SIFT-based layout similarity metric is introduced, then a clustering algorithm is performed to get representative layouts.

1) SIFT Layout Distance: SIFT [13] has been widely used in computer vision field to capture local features of an image. The output of SIFT is a set of key points that are captured through a staged filtering approach [13]. The key points are detected from different scales of DOG images by searching for the extreme points, and these extreme points are filtered so that the remaining points are stable enough. The detected key points are invariant to rotation, scaling and contain the main
gradient direction, so they can be chosen as the reference for classifying the layouts. There is an example of SIFT feature point distribution in Fig. 5. The white squares are patterns, and the red points stand for SIFT feature points. Local features are attached to the points, so they can be used to represent local point distribution in Fig. 5. The white squares are patterns, and the red points stand for SIFT feature points. Local features are attached to the points, so they can be used to represent local similarity of an image.

We measure the similarity of the two layouts by matching the feature points, as shown in Fig. 6. Let \( p, q \) be the 128-D feature vector calculated by the SIFT algorithm. \( D_{th} \) is the threshold to determine if the two feature points are matched. Therefore, the distance between two feature points (vectors) is defined as

\[
d(p, q) = \begin{cases} \sqrt{p^T q}, & \text{if } \sqrt{p^T q} \leq D_{th} \\ 1, & \text{otherwise} \end{cases}
\]

where \( p^T \) is transform of \( p \). Equation (10) shows that if the two feature points are close enough, the distance is the Euclidean distance. Otherwise, the distance between them is their L2-Norm which is 1. We set \( D_{th} \) to 0.7 in our implementation.

Algorithm 3 shows the similarity calculation between layout \( w \) and layout \( s \). For a feature point \( p_i^w \) in \( L_w \), we need to find an unmatched point \( p_j^s \) in \( L_s \) so that the distance \( d(p_i^w, p_j^s) \) between them is minimum. If the \( d(p_i^w, p_j^s) \) is less than \( D_{th} \), it means the similarity of these two points is high and the pattern distribution near this point is very similar, so we mark them as matched. Fig. 6 shows an example of SIFT matching, where the distance between two matched feature points is stored in \( D^{ws} \). Since the number of feature points differs as \( m \neq n \), we set

\[
D_{ws} = \min d(p_i^w, p_j^s) \quad \text{if } d(p_i^w, p_j^s) \leq D_{th}
\]

Algorithm 3 Calculate Layout Similarity

**Require:** Layout \( L_w \) and \( L_s \).
Let \( p_1^w, p_2^w, \ldots, p_m^w \) be the feature points in \( L_w \);
Let \( p_1^s, p_2^s, \ldots, p_n^s \) be the feature points in \( L_s \);
Initialize the empty array \( D^{ws} \);
for \( i = 1 \rightarrow n \) do
for \( j = 1 \rightarrow m \) do
if \( d(p_i^w, p_j^s) \leq D_{th} \) then
mark \( p_i^w, p_j^s \) matched;
put \( d(p_i^w, p_j^s) \) in \( D^{ws} \);
else
put 1 in \( D^{ws} \);
end if
end for
end for
sort \( D^{ws} \) in an ascending order;
\( S_{sift}(L_w, L_s) = \sum_{k=1}^{C} D^{ws}_k \),
return \( S_{sift}(L_w, L_s) \);

which is defined as

\[
S_{sift}(L_w, L_s) = \sum_{k=1}^{C} D^{ws}_k.
\]  

In our implementation, \( C \) is set to 60.

2) Clustering: Inputting a set of layouts, an adjacency matrix representing the layout distance can be calculated according to (11). Considering the mismatching of feature points will introduce distance noises, which greatly affect the performance of k-means, so we select the k-medoids as the clustering method. The central point of k-means is not a real point in the cluster and is calculated by all points, so noise points also contribute to the movement of central points. But k-medoids chooses real points as the central points, and they are selected by the sum distance in their clusters, therefore, the k-medoids is less sensitive to the noises as you can imagine. The performance of k-medoids is evaluated by the sum of layout distance (SLD), which is given by

\[
SLD = \sum_{i=1}^{M} \sum_{L_k \in C_i} S_{sift}(L_k, L_i)
\]

where \( C_i \) is a class whose center point is \( L_i \), while \( L_k \) is a noncentral point belongs to \( C_i \). There are \( M \) classes in total. SLD represents the sum of the distance from each noncentral point to its respective center point. The initial central points are randomly selected. The object is to reduce SLD iteratively by changing the center point and calculating the distance of each class. We set \( M \) to 50, and randomly select five layouts in each cluster.

B. Enhanced Layout Sampling

Although with the SIFT distance metric, our framework is able to obtain a fairly good performance which has been verified by our experiments, but due to the limitations of algorithm characteristics, there are still the following challenges.

1) The SIFT distance is based on the matching of similar points. Since SIFT only captures the feature points that are invariant to rotation or scaling, when most parts
of the image are relatively smooth, SIFT may not be able to catch enough feature points. In other words, when extracting the features on a layout composed of simple shapes, a modicum of feature points may cause the degradation of algorithm performance. For example, when matching points and calculating the distance, more feature points in Fig. 7(a) are better to describe the layout than that in Fig. 7(b).

2) Most of the SIFT feature points appear on the corners to detail the shapes. However, there are a lot of repeated shapes in a layout, which not only increases the calculation complexity but also causes over-characterization of a certain area where most of the feature points are distributed. This results in an unbalanced distance measurement among different subareas of a layout.

To address the above issues, we propose an alternative layout distance metric based on autoencoder. Autoencoders are a set of special neural networks used for extracting the latent representation, which contains two components: 1) encoder and 2) decoder. Let \( f(x)_e \) and \( f(x)_d \) denote encoder and decoder, respectively, autoencoder is trying to make the input \( x \in \mathbb{R}^n \) to be the same as the output, as shown in Fig. 8.

Encoder converts input into latent space expression which can be described as \( f(x)_e = y \in \mathbb{R}^m \). While decoder reconstructs the input with the process of \( f(y)_d = \hat{x} \in \mathbb{R}^n \). If we set \( m < n \), the training step of the autoencoder forces it to extract the main features of the image. The output \( \hat{x} \) is usually a blurred image because encoding is a lossy process that keeps the important image representation in latent space and drops unnecessary information. When training the autoencoder, we minimize the loss between the original image and the decoder output iteratively to find the best representation. Ideally, each parameter in the latent space \( \mathbb{R}^m \) is an independent variable that controls some properties of layout images. Based on above discussions, the layout distance computed from the extracted features is defined as follows:

\[
S_{\text{auto}}(L_w, L_s) = \|f(L_w)_e - f(L_s)_e\|_2
\]

where \( L_w \) and \( L_s \) represent different layouts. The distance between these two layouts is the \( L_2 \) norm of encoder output. Based on the distance definition, \( k \)-medoids algorithm introduced in Section IV-A can divide them into different clusters. A simple clustering result based on different distance metrics can be seen in Fig. 9. SIFT approach clusters the left and the middle layouts, but the autoencoder approach clusters the middle and the right layouts, which look much more similar. Considering the number of patterns and pattern density differ from each other, the degradation of the SIFT algorithm and the unbalanced measurements of layout may cause the misclassification using the SIFT approach.

The encoder structure resembles ResNet18 [24] with fully connected layers excluded and the decoder uses corresponding deconvolution layers to reconstruct the image. The encoder structure information can be seen in Table II, where “Layer” shows the layer type. Columns “Filter” and “Stride” represent the size of convolution kernels and their strides, and “Output” is the output shape of each layer. The encoder structure is mainly composed of basic blocks. There are two convolutional kernels in a basic block, and batch normalization is added to the end of each convolutional kernel. Each layer contains an identity mapping (it is not shown in the table) to enable the network to go deeper. Each entry in a two-element vector of “Stride” is the corresponding convolution kernel strides of a basic block. From the table, we can see there are 7 \( \times \) 7 \( \times \) 512 = 25 088 parameters to describe a layout, which is about half of the input image size. Table III lists the configurations of the decoder. It is a reversed structure of the encoder, and the deconvolution layer is used for the upsampling process to restore the image. The autoencoder implementation is based on PyTorch [25].

C. Decomposition Sampling

Similar to the problem of layout sampling, although traditional layout decomposition prunes many illegal decomposition results, the remaining decomposition choices still can be very large. The number of decomposition results increases exponentially with the number of patterns as illustrated in
Fig. 10. As the increase of about 9 patterns, the decomposition result size grows from less than 100 to more than 6000. Due to the limitations of computing resources, we are motivated to explore an effective decomposition sampling method. Considering the EPE occurrence is highly related to nearby pattern distribution, the complete combination of patterns in a subregion will be more helpful for our training and \( n \)-wise method is able to tackle this problem. In order to focus more promising decomposition solution, we also combine no-odd graph result and \( n \)-wise method to generate training set.

Different from the decomposition generation phase in Section III-A, here we divide patterns into two types to reduce the decomposition number, because generating the score of decomposition is much more time-consuming than predicting. According to (14), the patterns \( T \) with the distance \( d \) less than \( n_{\text{min}} \) are divided into \( S_P \), while the remaining pattern are in \( R_P \) set

\[
T \in \begin{cases} 
S_P, & \text{if } d \leq n_{\text{min}} \\
R_P, & \text{if } n_{\text{min}} < d 
\end{cases}
\]  

(14)

where \( n_{\text{min}} \) is set to 80 nm, that is same to the configuration in Section III-A.

Similar to generating \( \text{mergedArrs1} \) in Section III-A, we first divide patterns into two types, \( S_P \) and \( R_P \). By solving nonodd graph problem, we can obtain the relative position relationship of \( S_P \), then we build the three-wise arrays together with \( R_P \). Finally, we reverse the value and merge the same rows. In our implementation, generating three-wise arrays is a tradeoff between prediction accuracy and layout score simulation running time. Three-wise sampling strategy ensures that the training set contains the complete combination of any subregion with three patterns (part of patterns in \( S_P \) are excluded).

### D. Model Training

To avoid the dual layout problem, the input will be formatted as mentioned in Section III-A. In this article, \( L^2 \) error and EPE numbers are selected as the evaluation metrics. Since the occurrences of print violations will lead to a significant decline in printability, the score of decomposition is organized as follows:

\[
\text{score} = \alpha \times \#L^2\text{ Error} + \beta \times \#\text{EPE} + \gamma \times \#\text{Violation}. 
\]  

(15)

In our implementation, \( \alpha, \beta, \) and \( \gamma \) are 1, 3500, and 8000, respectively, and \( z \)-score regularization is applied which is given by

\[
\text{z-score} = \frac{\text{score}_i - \mu}{\sigma} 
\]  

(16)

where \( \text{score}_i \) is the \( i \)th layout score we defined in (15), \( \mu \) is the mean of decomposition results, and \( \sigma \) is the standard deviation. It describes how many standard deviations the score is from the mean, and data are converted into the same magnitude to ensure the comparability between the data. We use \( z \)-score as the decomposition quality. The mean absolute error (MAE) is applied as the cost function

\[
\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} 
\]  

(17)

where \( y_i \) denotes the label of the \( i \)th decomposition in the training set, and \( \hat{y}_i \) is the corresponding predicted value.

The image dimension is \( 224 \times 224 \), which increases the complexity of training. Here, Adam optimizer [29] is selected to train the model. Compared to the mini-batch gradient, Adam computes individual adaptive learning rates for different parameters which is more suitable for large-scale data.

We also take the structure of ResNet18 as the basic regression network (see Fig. 11). It is similar to the encoder structure, and details of convolutional layers can be seen in Table II. The network implementation is based on PyTorch [25] library. The identity mapping in each basic block enables the net layer to become deeper to obtain a better regression of object function. The input of the net is \( 224 \times 224 \times 1 \) tensor to receive a grayscale image, and the output of stacked conventional layers is \( 7 \times 7 \times 512 \) cube. After average pooling, there is a 1000 dimensions layer, and a fully connected layer is added to output the score.
V. EXPERIMENTAL RESULT

The proposed framework is implemented in C++ and validations are performed on Intel i7 3.6-GHz CPU. The printability estimation network and autoencoder are implemented in PyTorch [25]. To generate \( n \)-wise sampling arrays, we use PICT [30], an open source C library. The open source lithography simulator and EPE checker are from [19]. The EPE violation threshold is set to 10 nm and the approximated optical model kernel number \( K \) is set to 24. Experiments are conducted on an open source cell library NanGate [31].

The layout dataset is generated using [32], which takes a set of design rules for contact layer and yields in total 8000 designs. These designs resemble NanGate 45 nm library and are verified with Mentor Calibre design rule check.

A. Framework Evaluation

In the first experiment, we compare the optimization result of our framework with previous unified framework and two-stage independent flow on standard cell library NanGate, as shown in Table IV. We obtain binary from the authors of [9], and the results of the two other conventional flows are directly from [9]. Columns “EPE” and “Time (s)” list the number of EPE violations and the time elapsed in seconds when the optimization is convergent. Column “Ours” lists the results of our deep learning-based framework without applying density balance (in Section III-B), while column “Ours-Balanced” lists the results with the density balance checking. Compared with previous work [28], our frameworks (“Ours” and “Ours-Balanced”) use autoencoder as distance metric.

In Table IV, our frameworks show significant improvement in “EPE” and “Time” on average. The runtime of “Ours” is 239.10 s, which achieves around \( 4 \times \) speed-up compared with [26]+[5] and [27]+[5], and \( 8 \times \) speed-up compared to “[9].” That is because our framework does not need to get the decomposition by solving the SDP problem [9] and select decomposition candidates by lithography simulation, both of which are extremely time consuming.

In terms of “EPE,” “Ours” approach reduced five EPE violations compared with [28] in total. The EPE violation performance between [28] and “Ours” is very similar in most cases, except for NOR3_X2 test case (four EPE violations versus one EPE violation). One possible reason why [28] results in a greater number of EPE violations (four EPE violations in cell NOR3_X2) is that the pattern distribution of this layout is unfriendly for ILT optimization. But it is encouraging to see the training set of “Ours” reaches better coverage of layout data, so it can handle this kind of layout distribution well.

There are some examples of optimization results in Fig. 12. The EPE violations are marked as red crosses in the image. It can be seen that both of SIFT approach and autoencoder approach have fewer EPE violations and better printability on contact layers of the standard cells.

<table>
<thead>
<tr>
<th>ID</th>
<th>[26] + [5]</th>
<th>[27] + [5]</th>
<th>[9]</th>
<th>SIFT distance [28]</th>
<th>Ours-Balanced</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EPE</td>
<td>Time (s)</td>
<td>EPE</td>
<td>Time (s)</td>
<td>EPE</td>
<td>Time (s)</td>
</tr>
<tr>
<td>INV_X1</td>
<td>1</td>
<td>1183</td>
<td>1</td>
<td>1183</td>
<td>1</td>
<td>1995</td>
</tr>
<tr>
<td>NOR2_X1</td>
<td>8</td>
<td>1421</td>
<td>4</td>
<td>1405</td>
<td>1</td>
<td>1996</td>
</tr>
<tr>
<td>BUF_X1</td>
<td>5</td>
<td>867</td>
<td>5</td>
<td>1068</td>
<td>1</td>
<td>1990</td>
</tr>
<tr>
<td>CLEUBF_X1</td>
<td>1</td>
<td>1046</td>
<td>1</td>
<td>737</td>
<td>0</td>
<td>1996</td>
</tr>
<tr>
<td>OR1211_X1</td>
<td>5</td>
<td>1213</td>
<td>7</td>
<td>1207</td>
<td>6</td>
<td>1996</td>
</tr>
<tr>
<td>AG1211_X1</td>
<td>5</td>
<td>1048</td>
<td>8</td>
<td>1080</td>
<td>1</td>
<td>1989</td>
</tr>
<tr>
<td>AND2_X1</td>
<td>13</td>
<td>1080</td>
<td>13</td>
<td>1061</td>
<td>1</td>
<td>1993</td>
</tr>
<tr>
<td>OR2_X1</td>
<td>7</td>
<td>1046</td>
<td>3</td>
<td>1220</td>
<td>0</td>
<td>1997</td>
</tr>
<tr>
<td>NAND4_X1</td>
<td>5</td>
<td>1175</td>
<td>6</td>
<td>1176</td>
<td>1</td>
<td>1997</td>
</tr>
<tr>
<td>NAND2_X1</td>
<td>7</td>
<td>1175</td>
<td>8</td>
<td>774</td>
<td>1</td>
<td>1998</td>
</tr>
<tr>
<td>OR4_X1</td>
<td>5</td>
<td>1188</td>
<td>2</td>
<td>1233</td>
<td>0</td>
<td>1987</td>
</tr>
<tr>
<td>NOR3_X2</td>
<td>5</td>
<td>773</td>
<td>4</td>
<td>1168</td>
<td>7</td>
<td>1999</td>
</tr>
<tr>
<td>OA133_X1</td>
<td>11</td>
<td>1017</td>
<td>10</td>
<td>922</td>
<td>6</td>
<td>1998</td>
</tr>
</tbody>
</table>

Average: 600.00 1094.31 | 5.53 1094.92 | 2.15 1994.69 | 0.76 241.84 | 0.69 247.80 | 0.39 239.10 |
Ratio: 15.39 4.58 | 14.18 4.58 | 5.51 8.34 | 1.94 1.01 | 1.77 1.03 | 1.00 1.00 |

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 20,2022 at 01:58:36 UTC from IEEE Xplore. Restrictions apply.
Fig. 12. Comparison with ICCAD’17 [9] and DAC’20 [28] on: (a) BUF_X1; (b) OAI211_X1; (c) NAND3_X2. (d) NOR3_X2. (e) OAI33_X1. Compared with the state of the art, in all five cases, our framework results in fewer EPEs.

B. Sampling Strategy Evaluation

To demonstrate the efficiency of our sampling strategy, we compare the EPE number of the random sampling approach and the proposed approach. As shown in Fig. 13, “SIFT distance” and “Autoencoder distance” are trained in the way we introduced in Section IV. “Random Sampling” is trained in the same way but the training set is randomly sampled from generated layouts and layout decompositions are randomly generated from the corresponding sampled layouts. Although these layouts and decompositions in “Random” are randomly sampled, the decomposition generation module still offers legal decomposition results to the estimation module. It can be observed in the bar chart that the EPE number of “Random sampling” strategy is more than three times of “Autoencoder distance” strategy and twice of “SIFT distance” strategy. Since random sampling layouts can hardly find promising results in solution space, the training set distribution should obey the distribution of generated layouts. As a result, random sampling quality is determined, to some extent, by the quality of distribution of the generated layout database, which may not promise a reasonable layout sampling result.

For a specific layout, the solution space is too large, but many of the results can be discarded by formulating some physical rules, so random sampling needs to sample more decompositions to fill the gap. Our sampling strategy focuses on the possible decomposition while the random sampling is equivalent to sampling evenly at the solution space, which needs much more training data. An example of a printed image optimized by the three different estimation models is illustrated in Fig. 14.
C. Individual Printability Estimation

In the LELE-type lithography setting, the printability can be estimated independently, through which two decomposed mask scores can be estimated individually and summed up. In the training stage, we can generate decomposition candidates as two images. The forward optimization flow is the same as our framework, shown in the left part of Fig. 2, except for the printability prediction module. The new estimation model has two advantages.

1) It is a more flexible method, as a well-trained model can be extended to triple mask patterning technology and beyond without retraining the model.

2) This kind of model will not meet the dual problem represented as Fig. 4, so it is an easier way to process data.

We implemented this model and conducted the experiments, and the corresponding results are shown in Fig. 15. The new estimation model is represented with yellow color, while “Ours” is the approach introduced in Section III. From the result we can observe that the “Simplified model” shows a good result with eight EPE violations, which is fewer than [28], but still has three more EPE violations than “Ours.”

D. Handling Large-Scale Chips

The proposed method is verified on standard cells, and it is possible to extend our framework to the layouts of large circuits. In order to show the scalability of our framework, we conducted the experiment on a large circuit (10840 nm × 7890 nm) synthesized using the same NanGate [31] library as used in previous experiments. The layout is divided into 22 clips using a sliding window-based method with overlap, in which some empty and duplicated clips are ignored in the result. The results are shown in Table V. We implemented the decomposition generation and selection method of LDMO [9] as the baseline, the results are listed in the “LDMO [9]” column of Table V. “Ours” is the introduced framework without applying density balance checking. The average EPE violations of “Ours” are 2.09, while “LDMO [9]” outputs 2.68. Although very dense or very sparse pattern distributions may lead to the degradation of our decomposition generation, “Ours” still reduces 22% EPE violations and takes only 36% runtime compared to the baseline. The printability prediction module shows its effectiveness and efficiency when handling divergent layouts. The large circuit design and some clip optimization results are illustrated in Fig. 16.

VI. DISCUSSION

A. Adopting ML-Based Lithography Simulation

In addition to the machine learning (ML)-based printability prediction method we used in our framework, ML-based lithography simulation methods [33], [34] are also widely applied. The main differences are that, given some masks, ML-based printability prediction tells if the...
masks/decomposition candidates are OPC-friendly through inferring some concrete evaluation metrics (e.g., EPE# and L2 error in this work). While ML-based lithography simulation outputs the printed shapes, which can be used to infer the performance on different metrics. It usually faces a much larger solution space. But please note that it cannot estimate the printability after many iterations of OPC, as this kind of model just accelerates the lithography simulation process, and the output changes with the reshaping of masks. In general, the two methods focus on different points and can be applied for different purposes.

In our framework, obtaining the optimal MO result can be divided into two steps: 1) finding the best decomposition and 2) conducting ILT for this pair of masks. In order to find the best decomposition, we applied the ML-based printability prediction method. As for the subsequent ILT process, ML-based lithography simulation can be chosen as an acceleration method.

B. Handling More Masks

The related modules for applying more advanced MPL are decomposition generation module and ILT optimization module. For decomposition generation module, it mainly involves two processes: 1) building the no-odd graph and 2) applying n-wise method. In the scenario of triple patterning lithography, decomposition generation module solves three-coloring problems, where assigning different colors to the neighbor in no-odd graph cannot handle. But methods for triple patterning decomposition have been well investigated [1]–[4]. By adopting [3], we can solve the conflicts among violated patterns. As for n-wise method, it is suitable for the three-coloring problem because it can tackle a factor with three levels, where each level can be viewed as a color. Then applying (three-level) n-wise method to generate arrays and combining the results, we are able to obtain three-color decomposition candidates.

For ILT optimization module, we can observe from (4) and (5) that the optimization formulations are not limited to the mask number. Therefore, it can be easily extended by introducing a new mask variable $M_3$. After applying the sigmoid transformation to $M_3$, ILT is able to minimize the $L_2$ error, thus our framework can handle more advanced MPL.

C. Extend to Other Layers

Our framework is developed based on contact layers, but it has the potential to be extended to metal layers. Since the printability prediction module is not limited by a particular layer, the prediction model still can supervise the layout decomposition process and is expected to achieve excellent performance on other layers with sufficient training data. As for the decomposition generation module, a much denser conflict graph may cause the degradation on decomposition generation performance and stitch insertion problems need to be handled. Therefore, more general decomposition algorithms [1]–[4] that take stitch insertion into consideration, need to be applied for obtaining more proper decomposed masks.

VII. CONCLUSION

In this article, we propose a deep learning-driven framework to not just bridge the gap between LDMO but also speed up the procedure of selecting decomposition process. To improve the prediction accuracy and accelerate the training process, we use SIFT and autoencoder to extract layout features. K-medoids clustering and $n$-wise method are adopted to generate the training set. We also combine the no-odd graph and $n$-wise method to generate decomposition candidates with different pattern combination strengths. Experiments demonstrate that our framework can efficiently reduce EPE violations and accelerate the overall optimization process.

REFERENCES
