AdaOPC: A Self-Adaptive Mask Optimization Framework For Real Design Patterns

Wenqian Zhao
CUHK

Xufeng Yao
CUHK

Ziyang Yu
CUHK

Guojin Chen
CUHK

Yuzhe Ma
HKUST(GZ)

Bei Yu
CUHK

Martin D.F. Wong
CUHK

ABSTRACT

Optical proximity correction (OPC) is a widely-used resolution enhancement technique (RET) for printability optimization. Recently, rigorous numerical optimization and fast machine learning are the research focus of OPC in both academia and industry, each of which complements the other in terms of robustness or efficiency. We inspect the pattern distribution on a design layer and find that different sub-regions have different pattern complexity. Besides, we also find that many patterns repetitively appear in the design layout, and these patterns may possibly share optimized masks. We exploit these properties and propose a self-adaptive OPC framework to improve efficiency. Firstly we choose different OPC solvers adaptively for patterns of different complexity from an extensible solver pool to reach a speed/accuracy co-optimization. Apart from that, we prove the feasibility of reusing optimized masks for repeated patterns and hence, build a graph-based dynamic pattern library reusing stored masks to further speed up the OPC flow. Experimental results show that our framework achieves substantial improvement in both performance and efficiency.

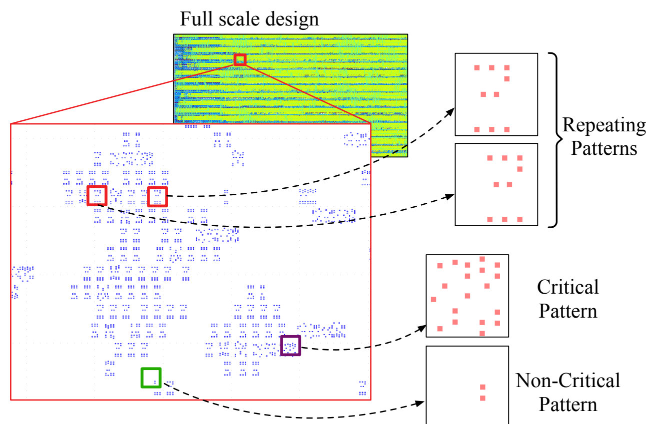

Figure 1: Visualization of a real design layer. Two discoveries motivated our OPC framework design: 1. Patterns scattered unevenly along the design layout with different complexity. We denote complicated patterns as critical and simple patterns as non-critical. 2. Patterns have large ratio of repetition on a full layout.

1 INTRODUCTION

Over the past few decades, VLSI technology node has been continuously shrinking, resulting non-neglectable lithography proximity effect, which affects the real manufacturability [1]. Resolution enhancement techniques (RETs) are utilized to improve the printability in the lithography process. Optical Proximity Correction (OPC) is one of the widely used RETs to optimize mask printability by compensating for the diffraction effect in the lithography process.

OPC approaches can be categorized into: (1) rule-based OPC [2], (2) model-based OPC [3–5], (3) inverse lithography technique (ILT)-based OPC [6–8] and (4) machine learning (ML)-based OPC [9–12]. Rule-based methods solve the problem heuristically, which is simple and fast but only suitable for less aggressive designs. Model-based OPCs mathematically model the lithography process and move/shift the edge fractures accordingly, ensuring mask fidelity but restricted by the solutions space in advanced technology nodes. ILT-based methods solve the inverse problem of the imaging system through optimizing an objective function, which is the most performant analytical method to tackle the OPC problem. As recent years witnessed the rapid development of machine learning algorithms and hardware, ML-based OPCs have shown remarkable speed-up in the OPC flows and are prevailing in design for manufacturing (DFM) academia. [9] and [13] use deep learning models for initial mask generation to reduce the iterations number of ILT. [14] uses a deep learning model to simulate the conventional ILT correction process. A lethal drawback is that machine learning model is a data-driven black box. Such methods are not guaranteed to work for some critical patterns. In summary, no approach is flawless and shows absolute superiority over others. Patterns with different complexity require different approaches.

Achieving desired OPC results with high efficiency on real designs require a systematic analysis on pattern distribution and complexity. By inspecting a real design, we come up with a few properties that can be leveraged to assist the analysis. Firstly, we notice that there exists a *diversity* of pattern density, which indicates that some regions are dense while some regions are sparse, as visualized in Figure 1, which implicates different kinds of OPC solutions are required. Moreover, as we take a closer observation of each sub-region, there also exists a *similarity* of the pattern distribution in different sub-regions. Many patterns are repeatedly placed on the full design layer. These patterns have similar geometric shapes and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICCAD '22, October 30-November 3, 2022, Gainesville, FL, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549468

are placed at different locations. Such pattern repetition enables us to utilize their shared geometric characteristics, which motivates the idea that the OPC solution of a pattern can be reused in a similar pattern for efficiency improvement. Motivated by these observations, we propose a self-adaptive framework, namely AdaOPC, for conducting OPC on real designs.

Firstly, AdaOPC is equipped with pattern analysis that can classify a sub-region as critical or non-critical, such that the proper OPC solver can be selected. More specifically, for densely scattered sub-regions, not only the diffraction effect but also the optical interference of incident light caused by neighboring components will jointly affect the final printed image. Such complex patterns for OPC are regarded as critical and more suitable for robust yet rigorous numerical optimization methods for higher manufacturability. In contrast, sub-regions with sparsely scattered patterns are simpler, and the mask optimization process is more suitable for machine learning models to learn with superior inference speed.

Secondly, given that many patterns are repetitive on the design layer with the same geometric shape, as shown in Figure 1, we investigate the feasibility of reusing the optimized masks of repeating patterns to avoid redundant iterations of OPC. There are three obstacles standing in the way of our idea: (1) As shown in Figure 2, slicing the large design layout into small patterns inevitably result in a location shift of patterns with the same geometric shape. Whether and how can an optimized mask with location shift be reused? (2) Given the query pattern, how to match a same pattern from a large number of stored ones accurately within an acceptable time? (3) How to measure the geometric similarity of patterns with location shift? In response to the mentioned three questions, we build a **dynamic pattern library** with online updating to store and reuse repeating patterns and optimized masks by constructing a dynamic hierarchical graph. We mathematically prove the shift equivariance property of the lithography process to show the feasibility of mask reuse by calculating the shift of design pattern and calibrating the mask. A graph-based approximation nearest neighbor search for pattern matching within a short query time. We summarize the contributions of this paper as follows:

- We propose a self-adaptive workflow for flexible OPC solver selection.
- We prove the feasibility of mask resue to speed up the OPC process for real design patterns and provide an efficient mask shift calibration method in practice.
- We generate design patterns embedding by supervised contrastive learning for similarity measurement and pattern matching.
- We construct a dynamic pattern library using a hierarchical graph with online update along with a greedy graph-based nearest neighbor search for fast matching.
- With experiments on different pattern cases from a real design layout, we proved our framework can reduce over 90% runtime while still preserving the optimal OPC performance.

2 PRELIMINARIES

2.1 Lithography Simulation Model

During the lithography process, an input mask **M** is projected through layers of optical lens onto a wafer plane. The intensity

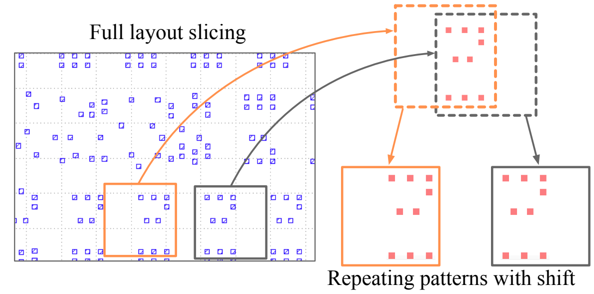

Figure 2: Slicing repeating full layout inevitably causes some location shift on repeating patterns.

after optical system **I**, namely the aerial image, leaves a coating on the wafer with photoresist to form the resulting pattern **Z**. The conventional simulation of the lithography process is composed of 2 consecutive components: optical projection model and photoresist model.

For the projection process, Hopkins diffraction model [15] has been widely used to analyze coherent imaging system mathematically. To avoid the computation complexity of the Hopkins model, a singular value decomposition model (SVD)-based approximation has been proposed by [16] and became the mainstream fashion. In the SVD model, the Hopkins diffraction model can be decomposed into a sum of coherent systems based on eigenvalue decomposition:

$$\mathbf{I}(x,y) = \sum_{k=1}^{N^2} w_k |\mathbf{M}(x,y) \otimes h_k(x,y)|^2, \quad x, y = 1, 2, ...N \tag{1}$$

where h_k is the k-th kernel and w_k is the corresponding weight of the coherent system. "$\otimes$" denotes the convolution operator. [17] indicates the K-th order approximation:

$$\mathbf{I}(x,y) \approx \sum_{k=1}^{K} w_k |\mathbf{M}(x,y) \otimes h_k(x,y)|^2, \tag{2}$$

We pick $K = 24$ in our experiment. After optical simulation, the lithography intensity **I** is sent to the photoresist model to generate the final binary pattern **Z** with an exposure resist threshold I_{th}:

$$\mathbf{Z}(x,y) = \begin{cases} 1, & \text{if} \quad \mathbf{I}(x,y) \geq I_{th}, \\ 0, & \text{if} \quad \mathbf{I}(x,y) < I_{th}, \end{cases} \tag{3}$$

Several machine learning-based lithography simulation methods have been proposed. [18] utilized a CNN network to perform a function model determination for resist model simulation. [19] developed a GAN-based LithoGAN, to map the input mask and output resist pattern. [20] proposed a two-stage DNN-based framework, solving the mask-to-SEM prediction as a domain-transfer problem and using CycleGAN [21] to learn the transferring process.

Although DNN models usually have the comparative speed advantage, we choose the Hopkins model for the reason of analyzability. A white box model enables us to analyze the pattern shift equivariance property mathematically during the lithography process.

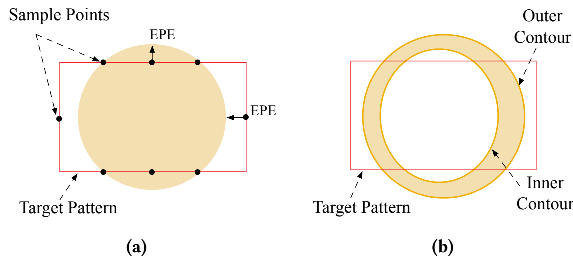

Figure 3: OPC evluation creteria: (a) Visualization of EPE measurement (b) Visualization of PVBand.

2.2 OPC Evaluation Criteria

2.2.1 Edge placement error (EPE).. After the lithography process, the printed image on the wafer has an inevitable geometric distortion from the design target. Edge placement error (EPE) is a common criterion to quantify distortion level. Measurement of EPE is visualized in Figure 3(a): A series of measuring points are sampled along the boundary of the target design pattern, including vertical edges and horizontal edges. If the distance D between printed image and target is larger than threshold th_{EPE} at a sample point, we label it as a EPE violation.

$$EPE_violation(x, y) = \begin{cases} 1, & D(x, y) \geq th_{EPE}, \\ 0, & D(x, y) \leq th_{EPE}, \end{cases} \tag{4}$$

2.2.2 Process Variation Band (PV Band). In real lithography applications, process variation may cause deviation in the final printed images, which possibly leads to printing failure. Given different lithography conditions such as focus/defocus depth and incident light intensity, printed images have various contour results. Process Variation Band (PV Band) is defined as discrepant (XOR) region of innermost and outermost contours as shown in Figure 3(b) to evaluate printing robustness.

$$PVBand = \sum_{x,y}^{N^2} |\mathbf{Z}_{out} - \mathbf{Z}_{in}|, \tag{5}$$

where N is the size of pattern. $\mathbf{Z}_{out}$ denotes the printed pattern of outer contour and $\mathbf{Z}_{in}$ denotes the inner contour.

3 ADAPTIVE FRAMEWORK

3.1 Workflow Overview

Our proposed workflow is visualized in Figure 4. In Section 3.2, we firstly introduce the extensible solver selection module to choose OPC solver for different patterns. In Section 4 we will demonstrate the dynamic graph-based Pattern library, one of the key components of AdaOPC framework for mask Optimization efficiency improvement. An approximate nearest neighbor searching(ANNS) method will be utilized to match similar patterns. Section 4.2 specified how we embed patterns into high dimensional vectors for pattern matching in the library using supervised contrastive learning. Section 5 will further discuss the mask reusability and requirement. A proof of the shift equivariance during lithography is provided to validate the feasibility, along with the solution by shift calibration.

3.2 Extensible OPC Solver Selection

Our framework maintains a flexible solver pool to select suitable OPC solutions for different patterns based on their complexity. We divide sliced design patterns into two groups: critical and non-critical patterns. Then we use a solver selector to choose which OPC solver to use. This Solver selector can be regarded as a 2-class classifier and built with a simple deep learning classification model. We use Resnet-18 [22] as backbone network with objective to minimize the cross-entropy loss L:

$$L = -\frac{1}{N} \sum_{i}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i), \tag{6}$$

where y_i is the 1/0 label for critical pattern of sample i, p_i is the probability predicted by classifier model. The model is trained to predict p_i as close as y_i. Without bells and whistles, such simple network + loss combination is capable of performing a fast and accurate prediction on pattern class.

For non-critical patterns, we build our ML-Solver using a generative neural network model, consistent with one of the SOTA ML-based OPC solvers DAMO-DMG [12]. We also pick U-Net++ with residual blocks inserted in the bottleneck as our model structure and train the generative model with the same strategy as [12]. The only difference is the training data. We prepare our own training dataset with patterns from a real full-scale design and mask generated by a robust OPC engine, of which the lithography model is an authentic one instead of a DNN simulator as in [12]. Such data preparation aligns with the real OPC scenario, where the only ground truth we have is the lithography model.

For critical patterns, we use rigorous numerical optimization method as [17] with GPU acceleration by CUDA, despite that deep learning approaches already reached good performance for an overall evaluation on some test sets of patterns. Data-driven black-box deep learning model may learn to mimic and reverse the diffraction effect very well. Nevertheless, they might have difficulty dealing with optical interference of incident light caused by complex neighboring components. In that case, the rigorous numerical optimization solver provides an analytical solution regardless of the geometric complexity of patterns. Moreover, in a real OPC scenario with a new design and even possibly a new lithography engine, patterns and optimized masks generated by robust methods can be the dataset to train the machine learning model to adapt to new settings.

Note that the solver pool is extensible. Any powerful OPC solution with certain strengths for certain patterns holds the possibility to be imported as a replacement or complementary candidate. If more than two solvers are in the solver pool, we can simply modify the classifier loss:

$$L = -\frac{1}{N} \sum_{i}^{N} \sum_{c=1}^{C} y_{ic} \log(p_{ic}), \tag{7}$$

where C is the number of pattern classes, the same as the number of corresponding OPC solvers. y_{ic} is the 1/0 label for whether this pattern belongs to class c. In this way, we can simply transform the problem into a multi-classification case.

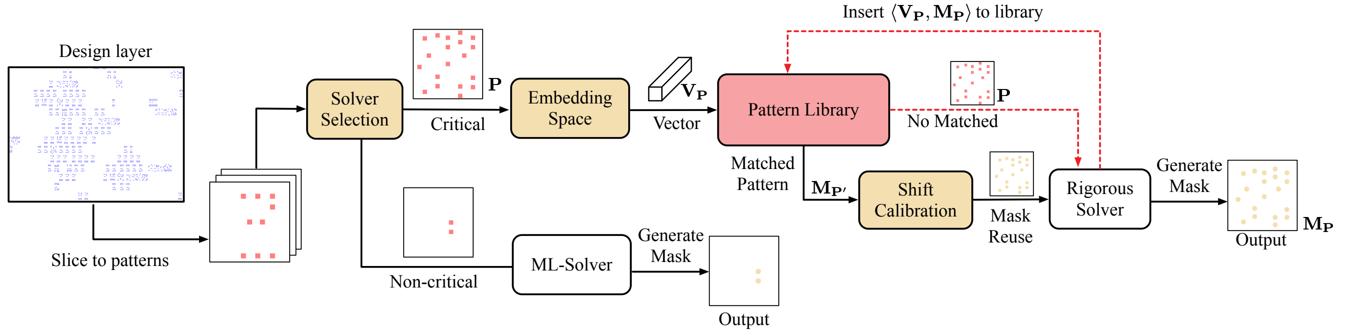

Figure 4: Overall workflow of AdaOPC. Colored blocks are functional modules. Red dashed lines represent library update flow.

4 DYNAMIC PATTERN LIBRARY

After filtering out simple cases, we need to apply a rigorous solver for critical patterns. For further efficiency improvement, we build a dynamic pattern library to store the pattern pairs: sliced pattern $\mathbf{P}$ and their corresponding post-OPC mask $\mathbf{M_P}$, hence to reuse the mask of repeating patterns to avoid the redundant time consumption of iterations-from-scratch OPC process. The key idea is to identify a stored repeating pattern before OPC. An online update mechanism enables a brand new pattern and corresponding mask to be inserted into the library.

We construct the pattern library with a graph structure as inspired by [23], where each node represents a pattern stored. Each edge connecting two nodes shows they are neighbors of each other with more similarity. Considering the number of patterns on a full design layout, the size of the graph is gigantic. Naive search for shortest distance requires pair-wise distance comparisons of nodes in the graph, which is impractical. We improve the matching time efficiency with:

- Sparse neighborhood graph structure, where nodes distant from each other are sparsely connected.
- Graph is divided into hierarchical layers. Nodes have restricted degree at each layer. more edges reside at lower layers, enabling greedy search of nearest neighbor at each layer.

The visualization of the hierarchical sparse graph structure is in Figure 5.

4.1 Pattern Matching And Online Update.

The task of matching pattern with the most similar geometric shape can be regarded as a nearest neighbor search problem (NNS). As inspired by [23], we utilize the Hierarchical Navigate Small World (HNSW) algorithm for fast matching. As shown in Figure 5, the pattern matching follows a greedy strategy to traverse the graph from higher layers to the bottom layer. A list of nearest pattern node candidates is maintained through the top-down traversal. The list is updated when a closer pattern appears during searching that has a distance shorter than one of the candidates. Such matching strategy is based on proximity graph nearest neighbors search. The detailed pattern matching search strategy at each hierarchical layer is illustrated in Algorithm 1.

After reaching the bottom layer, pattern candidates in list C with a distance smaller than a threshold σ are regarded as matched

Algorithm 1 Graph-Based Pattern Matching Greedy Search

```
1:  Input: query pattern P, starting nodes q_s, number of nearest neighbor
    to return k,layer number l, distance measurement d(·)
2:  Output: nearest pattern candidates C
3:  V ← q_s // Visited nodes
4:  W ← q_s // Waiting list of nodes to visit
5:  C ← q_s
6:  while |W| > 0 do
7:      q* ← nearest pattern from W to P
8:      q_f ← furthest pattern from C to P
9:      if d(P, q*) > d(P, q_f) then
10:         break
11:     end if
12:     for e ∈ neighbor(q*) in layer l do
13:         if e ∉ V then
14:             V ← V ∪ {e}
15:             q_f ← furtherest pattern from C to P
16:             if d(P, e) < d(P, q_f) or |C| < k then
17:                 W ← W ∪ {e}
18:                 C ← C ∪ {e}
19:                 if |C| > k then
20:                     remove furthest pattern from C to P
21:                 end if
22:             end if
23:         end if
24:     end for
25: end while
```

patterns. If the smallest distance in C is still larger than σ, we regard it as a new pattern. This approach remains fast and accurate even when the graph grows large with new patterns continuously being inserted in the library.

The pattern library updates in an online style. For every new pattern that comes with no matched ones, the mask is optimized with from-scratch OPC iterations. The library will insert the pattern and optimized mask as new node and update the edge hierarchy of the graph to store the pattern. Algorithm 2 shows the library online update step details.

As Algorithm 2 indicates, the new pattern will be inserted to one of the hierarchical layers with decaying probability. Edges will be added at the same layer between this pattern and top k nearest ones. As the neighbor nodes will have degree increase, an edge re-connection of these neighbors will be conducted once degree is above the upper bound k. Therefore, the degree of each node in the

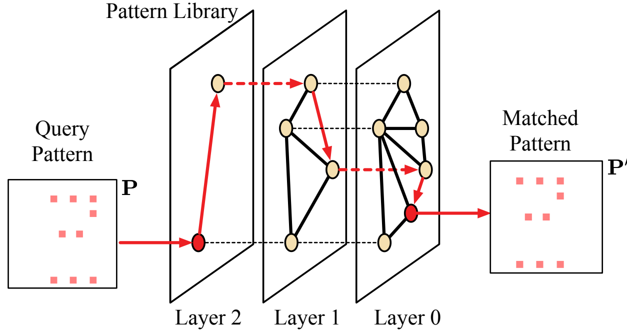

Figure 5: Visualization of the graph-based pattern matching flow. Query design pattern P greedily traversing the hierarchical graph. The nearest node reached at layer 0 corresponds to a match pattern P′ which has the most similar geometric shape with P.

Algorithm 2 New Pattern Insertion and Graph Update

1: **Input:**hierarchical graph G, new pattern $\mathbf{P}$, total layer number L, G's starting nodes q_s, max degree M
2: **Output:** updated hierarchical graph G
3: $l \leftarrow random(0, L)$ //exponentially decaying probability
4: **for** $l_c \leftarrow L, ..l$ **do**
5: $\quad C \leftarrow search(P, q_s, k, l_c)$ ▷ Algorithm 1
6: $\quad q_s \leftarrow$ nearest pattern of q in C
7: **end for**
8: **for** $l_c \leftarrow l, ..0$ **do**
9: $\quad$ insert $\mathbf{P}$ to layer l_c of G // add $\mathbf{P}$ into graph
10: $\quad$ C $\leftarrow search(P, q_s, k, l_c)$ ▷ Algorithm 1
11: $\quad neighbors((P)) \leftarrow$ top M nearest patterns in C
12: $\quad$ **for** $e \leftarrow neighbors(\mathbf{P})$ **do**
13: $\quad\quad$ add edge (P, e)
14: $\quad\quad$ **if** degree of $e > M$ **then**
15: $\quad\quad\quad neighbors(e) \leftarrow$ top k nearest patterns connecting e
16: $\quad\quad\quad$ remove all edges connecting e
17: $\quad\quad\quad$ create edges e with each one in $neighbors(e)$
18: $\quad\quad$ **end if**
19: $\quad$ **end for**
20: **end for**

graph is restricted by k. Note that the number of edges results in the matching complexity. Such a sparse hierarchical graph enables a fast search when the size of graph grows up.

Since vectors of the same pattern are closer to each other, we proposed some distance metrics to evaluate the similarity of vectors. The inner product of two vectors can be used to evaluate the direction difference of vectors:

$$Inner(\mathbf{V_{P_1}}, \mathbf{V_{P_2}}) = \mathbf{V_{P_1}} \cdot \mathbf{V_{P_2}} = \sum_{i=0}^{k} V_{\mathbf{P}_1,i} V_{\mathbf{P}_2,i}, \tag{8}$$

where $\mathbf{V_{P_1}}$ and $\mathbf{V_{P_2}}$ are the embedded vectors of pattern $\mathbf{P}_1$ and $\mathbf{P}_2$. $k = 256$ is the dimension of the embedded vector. Note that the inner product violates the positivity property where an element can be closer to some other element than to itself.

Consine similarity avoid the violation thus can be utilized to measure the similarity between two vectors of an inner product space:

$$\begin{aligned} d_{Cosine}(\mathbf{V_{P_1}}, \mathbf{V_{P_2}}) &= 1.0 - \frac{\mathbf{V_{P_1}} \cdot \mathbf{V_{P_2}}}{\|\mathbf{V_{P_1}}\| \|\mathbf{V_{P_2}}\|}, \\ &= 1.0 - \frac{\sum_{i=0}^{k} V_{\mathbf{P}_1,i} V_{\mathbf{P}_2,i}}{\sqrt{\sum_{i=0}^{k} V_{\mathbf{P}_1,i}^2} \sqrt{\sum_{i=0}^{k} V_{\mathbf{P}_2,i}^2}}, \end{aligned} \tag{9}$$

Euclidean Distance is another approach where the embedding metric space is regarded as a Euclidean Space. Each vector represents its position in the Cartesian coordinates. The similarity of two vectors can be evaluated directly by calculating the squared -2 norm of the difference of the coordinates:

$$d_{Euclid}(\mathbf{V_{P_1}}, \mathbf{V_{P_2}}) = \|\mathbf{V_{P_1}} - \mathbf{V_{P_2}}\|_2^2 = \sqrt{\sum_{i=0}^{k} (V_{\mathbf{P}_1,i} - V_{\mathbf{P}_2,i})^2}. \tag{10}$$

All metrics abide by the rules of nearest neighbor search (NNS) are feasible similarity measurement metrics, which leave possibility for exploration of various metrics according to different embedding space.

4.2 Embedding Space Construction

To reuse mask stored in library we need to match a pattern with same geometric shape. However it is not straighforward to compare geometric similarity of two patterns directly. We develop a embedding metric space which reflects the geometric property using a high-dimensional vector representation $\mathbf{V_P}$. Original $\langle \mathbf{P}, \mathbf{M_P} \rangle$ pair stored in library is replaced with $\langle \mathbf{V_P}, \mathbf{M_P} \rangle$ pair. In this way, a decision of whether two patterns are the same can be determination by a similarity metric of two embedded vectors.

We transform the embedding space construction into a feature extraction process by using deep learning model, and the metric space is built through deep metric learning. The embedded vector is the output of an embedding neural network. Deep learning model for the embedding process is composed of two modules:

- *Encoder*, $Enc(\cdot)$. For each input pattern $\mathbf{P}$, the encoder will encode the input pattern to a feature map $\mathbf{F}_P \in \mathbb{R}^{h \times w \times c}$, where h, w are the spatial size of the feature map $\mathbf{F}_P$ and c is the number of channels.
- *Projector*, $Proj(\cdot)$, which embeds feature map $\mathbf{F}_P$ to a representation vector $\mathbf{V_P} \in \mathbb{R}^k$. The output $Proj(\mathbf{F}_P)$ is normalized to the unit hypersphere in $\mathbb{R}^k$ at training stage for loss calculation.

Therefore, the embedding process is formulated as:

$$\mathbf{V_P} = Proj(Enc(\mathbf{P})) \in \mathbb{R}^k. \tag{11}$$

Previous deep learning-based OPC approaches [9, 12, 19] chose UNet or special variant UNet++ as the backbone structure. A harsh requirement of OPC problem strongly limits the selection of network backbone structure: the output mask will necessarily remain the same resolution as the input design.

Embedding process without such limitation leaves us the flexibility to more network structure candidates. We deliberately choose one of the most common structures: Resnet-18 [22] as encoder. Each input pattern $\mathbf{P}$ is a 2048 × 2048 2-D picture. In order to restrict the heavy computation caused time delay, we downsample

the pattern in a greedy manner until 256*256 before sending it into the neural network, without noticeable performance degradation. After the original Resnet-18 structure, an extra 1×1 convolution layer follows to shrink the feature channel size from 512 to 256. A linear layer is put at the end of the neural network to transform the 3-D feature into the final 1-D embedded vector $\mathbf{V_P}$. The size of $\mathbf{V_P}$ is a trade-off where larger size indicates higher matching accuracy but slower similarity computation and matching speed. We select 256 through experiments to guarantee good performance with neglectable matching time.

The embedding space $\mathbf{S}$ is specially designed with certain objectives: (1) patterns of the same shape share similar embedded vectors with the shortest distance. (2) patterns of different shapes clustered sparsely in the embedding space, far from each other. The training process of such embedding requires abundant data of different patterns as well as data of the same patterns. During training, the data from the same pattern is regarded as positive samples, and the embedded vectors will be pushed as close as possible with higher similarity, while the embedded vectors of different patterns are regarded as negative samples pushed as far as possible. We crop a great amount of patches of pattern from a real full-scale design for the training dataset.

Data Preparation. The cropping process has two steps to generate positive samples and negative samples as required by the training objective. The first step is to randomize some anchor points along the design layer. In the second step, with a random shift around each anchor point, we crop a certain number of patches, which will hold the same pattern within the square patch but with different relative positions. For each batch of training data, patterns will be labeled by their anchor point and thus abides by the positive/negative samples requirement.

Supervised Contrastive Loss. As we draw the picture of training the neural network to learn how to embed patterns to representative vectors, the traditional cross-entropy loss is not sufficiently sensitive to handle inter-class distance or noise labels. Among the family of losses based on metric distance learning [24–26], Contrastive loss [27] is one of the most powerful losses for learning representative embedding in the self-supervised learning domain. Inspired by [28], we extend the contrastive loss to supervised contrastive loss as all the positive/negative samples are genuinely generated and labeled at data preparation stage. As mentioned, representation vector $\mathbf{z}$ are normalized from $\mathbf{V_P}$:

$$\mathbf{z} = \text{normalize}(Proj(Enc(\mathbf{P}))) \in \mathbb{R}^k, \tag{12}$$

Then the loss function is formulated as:

$$\mathcal{L}_{supCon} = -\sum_{i \in I} \frac{1}{|J(i)|} \sum_{j \in J(i)} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_j / \tau)}{\sum_{a \in A(i)} \exp(\mathbf{z}_i \cdot \mathbf{z}_a / \tau)}, \tag{13}$$

where $i \in I$ is anchor indices of the training batch. $j \in J$ is the anchor indices of the positive samples. $A(i) = I \backslash \{i\}$ is all anchor indices except for i in this batch and therefore $A(i) \backslash \{J(i)\}$ are the anchor indices of negative samples. τ is a scalar temperature parameter. Term $\exp(\mathbf{z}_i \cdot \mathbf{z}_j / \tau)$ in numerator denotes similarity of positive sample pairs $\mathbf{z}_i$ and $\mathbf{z}_j$. $\exp(\mathbf{z}_i \cdot \mathbf{z}_a / \tau)$ in denominator denotes similarity of all sample pairs including negative ones. By minimizing the loss, the training process enlarges the similarity of positive samples and reduces the similarity of negative samples.

5 MASK REUSE WITH SHIFT CALIBRATION

5.1 Mask Reusability

We assume that repeating patterns can share mask for efficiency if the query design pattern $\mathbf{P}$ has a matched design pattern $\mathbf{P'}$ stored in the library with the same shape. However, Figure 2 has shown that when the whole design is sliced into small patterns, it is inevitable to find pattern location shift $(\Delta x, \Delta y)$ between $\mathbf{P}$ and $\mathbf{P'}$. In real lithography and OPC flow, if no surrounding factors affect lithography, the printed wafer image patch must have no distortion but only an identical shift to the design pattern, as visualized in Figure 6. Therefore, if we want to reuse the mask, the first requirement is to make sure that location shift during lithography will not cause any geometric distortion.

We mathematically prove the location shift remains unchanged before and after the lithography in order to show the feasibility of the mask shift calibration approach, we denote the Hopkins diffraction model through lithography in Section 2.1 as $Litho(\cdot)$ and the location shift as $\delta_{\Delta x, \Delta y}(\cdot)$, we show that:

Theorem 1 (Shift Equivariance)**.** *Given pattern* $\mathbf{P}$ *and mask* $\mathbf{M_P}$ *where*

$$\mathbf{P} = Litho(\mathbf{M_P}). \tag{14}$$

The following statement always holds:

$$\delta_{\Delta x, \Delta y}(\mathbf{P}) = Litho(\delta_{\Delta x, \Delta y}(\mathbf{M_P})). \tag{15}$$

Proof. For any position (x,y) on pattern $\mathbf{P}$:

$$\begin{aligned}
&\delta_{\Delta x, \Delta y}(\mathbf{P}(x,y)) = \mathbf{P}(x+\Delta x, y+\Delta y),\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x+\Delta x, y+\Delta y) \otimes \mathbf{M_P}(x+\Delta x, y+\Delta y) \right|^2,\\
&= \sum_{k=1}^{N^2} w_k \Big| \sum_{i=1}^{N} \sum_{j=1}^{N} h_k(i,j) \mathbf{M_P}(x+\Delta x+i-\frac{N}{2}, y+\Delta y+j-\frac{N}{2}) \Big|^2,\\
&= \sum_{k=1}^{N^2} w_k \Big| \sum_{i=1}^{N} \sum_{j=1}^{N} h_k(i,j) \mathbf{M_P}(x+i-\frac{N}{2}+\Delta x, y+j-\frac{N}{2}+\Delta y) \Big|^2,\\
&= \sum_{k=1}^{N^2} w_k \Big| \sum_{i=1}^{N} \sum_{j=1}^{N} h_k(i,j) \delta_{\Delta x, \Delta y}(\mathbf{M_P}(x+i-\frac{N}{2}, y+j-\frac{N}{2})) \Big|^2,\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x,y) \otimes \delta_{\Delta x, \Delta y}(\mathbf{M_P}(x,y)) \right|^2,\\
&= Litho(\delta_{\Delta x, \Delta y}(\mathbf{M_P}(x,y))).
\end{aligned} \tag{16}$$

Then Equation (15) is proved. □

Since mask shift will only result in a printing shift after lithography, repeating patterns in design can share OPC-optimized masks with a simple shift correction. We pick matched mask $\mathbf{M_{P'}}$ stored in pattern library and add correction $(-\Delta x, -\Delta y)$ to acquire the initial mask $\mathbf{M_P}$ for $\mathbf{P}$.

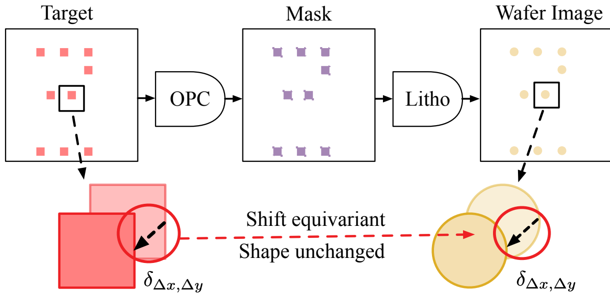

Figure 6: Printed wafer image must share identical location shift to design pattern with no geometric shape distortion.

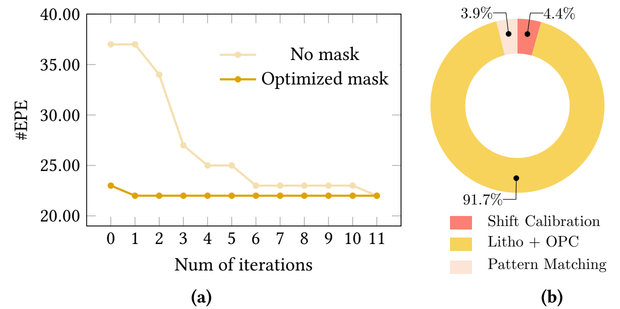

Figure 7: (a) EPE convergence comparison (b) Runtime breakdown of AdaOPC on critical patterns

Library Size	128-D	256-D	512-D
100	0.9ms	1.3ms	1.5ms
500	3.4ms	5.7ms	8.8ms
1000	9.0ms	13.3ms	22.2ms
2000	20.4ms	31.2ms	52.2ms
5000	59.8ms	93.0ms	156.6ms
10000	130.1ms	206.5ms	413.6ms

Table 1: Pattern matching speed Analysis on different embedding dimension.

5.2 Pattern Shift Calibration

We calculate the shift by computing the pixel-level similarity of two patterns $\mathbf{P}$ and $\mathbf{P}'$. The pixel-wise **cross-correlation** of $\mathbf{P}$ and $\mathbf{P}'$ reflects the pixel-wise similarity where the pixel of highest response value on the correlation map is the position shift of center point (x_{ctr}, y_{ctr}). The cross-correlation computation of two large 2-D pattern is time-comsuming. The calculation process of cross-correlation equal to convolution of $\mathbf{P}$ and $Rotate(\mathbf{P}')$, where $Rotate(\cdot)$ denotes rotation of 180°:

$$CrossCorr(\mathbf{P}, \mathbf{P}') = Conv(\mathbf{P}, Rotate(\mathbf{P}')), \tag{17}$$

We replace the calculation with Convolution and accelerate the computation with Fast Fourier Transform (FFT) [29]. The pattern shift can be calculated with:

$$\begin{aligned} x^*, y^* &= \operatorname*{argmax}_{x,y}\ Conv_FFT(\mathbf{P}, Rotate(\mathbf{P}')), \\ \Delta x &= x^* - x_{ctr},\ \Delta y = y^* - y_{ctr}, \end{aligned} \tag{18}$$

And the initial mask is corrected with:

$$\mathbf{M_P} = \delta_{-\Delta x, -\Delta y}(\mathbf{M_{P'}}). \tag{19}$$

In real application, we send calibrated mask into lithography model to verify the mask and into ILT solver for one or two further iterations if necessary, just in case of any noise caused by shift calibration operation. We use the same pattern size as [9] 2048 × 2048. With our implementation, the shift calculation time is less than 0.25s on CPU.

6 EXPERIMENTAL RESULTS

Our framework is mainly developed in Python. All machine learning modules in our framework are implemented using PyTorch. Lithography and ILT modules are developed in C/C++ with CUDA toolkit. All performance and speed experiments are conducted on CentOS-7 system with Intel i7-5930K 3.50GHz CPU and Nvidia GTX Titan X GPU. We choose the public lithography engine from ICCAD 2013 CAD Contest [30] with the 24 optical kernels. The photoresist intensity threshold is set at 0.055. We adopt lithography wavelength 193nm with defocus range of ±25nm and dose range of ±2%. EPE violation threshold th_{EPE} is set to 15nm.

In accordance with the authentic OPC scenario, all data used in our experiments is from a real design which we extracted from a gds file generated by the open-source layout generation tool OpenROAD [31]. We sliced patterns of size 2048×2048 in alignment with previous work [8, 9, 12] from a full scale via layer with more than 1.9×10^6 vias, where each pixel represents 1nm^2. For the training dataset of ML-Solver, we randomly slice 4000 patterns off the design layer with masks optimized by ILT Solver. We use the same patterns to train the pattern classifier. As for critical/non-critical labels, we directly apply lithography and label them by EPE number, which is intuitive as it reflects the mask optimization difficulty. For the training data of Metric Space embedding, we follow the steps in Section 4.2, using 400 random anchor points and shift around each anchor point within ±10% pattern width for 400 repeating patterns. The number of positive samples for each point should be equal to or larger than the number of anchors to guarantee a positive/negative ratio of each training batch.

6.1 Performance Analysis

In the begining, we validate the effectiveness of mask reuse by observing EPE descending trend. Firstly we present a demo experiment on a pattern, recording the EPE descending trend of the mask during ILT iterations with a calibrated optimized mask as the initial state. In comparison, we also record the trend with no initial mask and start the ILT process from scratch. As Figure 7(a) shows, EPE number with an initial mask starts at nearly optimal 23, and the descending trend converged at 1st iteration. In contrast, ILT from scratch gives 37 EPEs initially and takes six iterations to reach the initial EPE number with mask reuse, and overall 12 iterations to converge to 22.

To verify the efficiency of our framework, we experiment with the runtime analysis. Since non-critical patterns are handled by extremely fast machine learning-based methods, we mainly focus on critical patterns. Figure 7(b) visually demonstrates the time cost proportion of each step for critical pattern OPC in AdaOPC. We can see that 91.7% running time is spent on lithography and

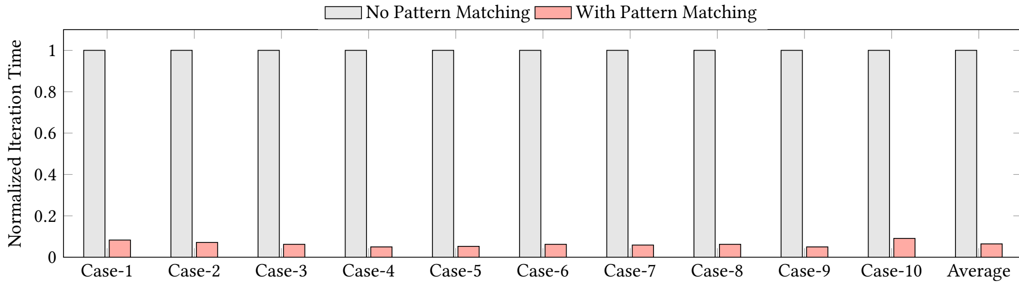

Figure 8: Mask convergence speed comparison with/without Pattern Matching. The time of mask optimization without pattern matching is normalized to 1 to demonstrate the acceleration ratio clearly. Optimization with pattern matching takes calibrated mask as the initial state.

Table 2: Comparisons of baseline approaches

Test Case	DAMO-DGS [12]			ILT-GPU [17]			AdaOPC		
ID	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)
1	22	23323	5.20	23	23329	41.15	22	23232	5.50
2	26	26729	5.26	25	26762	48.5	24	26580	5.41
3	27	26938	5.22	24	26720	55.92	24	26718	5.37
4	36	27975	5.18	29	28127	70.57	25	27934	5.40
5	35	28805	5.32	30	28925	66.89	30	28927	5.44
6	30	26960	5.31	25	26762	55.81	24	26775	5.38
7	33	26382	5.23	28	26453	59.47	28	26281	5.43
8	32	30646	5.38	25	29450	54.88	27	29341	5.42
9	25	24054	5.25	24	24053	70.62	23	24022	5.43
10	24	21939	5.29	23	21701	37.59	22	21644	5.53
Avg.	29.0	26375	**5.26**	25.6	26228	56.14	**24.9**	**26145**	5.43
Ratio	1.165	1.009	**0.970**	1.028	1.003	10.340	**1.000**	**1.000**	1.000

ILT OPC iterations while the time overhead of pattern matching and shift calibration combined is only 8.3%, which is trivial to the whole process. This also verifies the extensibility of our framework, leaving room for new powerful and fast OPC tools or litho-model to be imported for further speed up. Moreover, we also considered the condition when the pattern library gets larger. Although we cannot generate too many "ground-truth" masks due to time and computation resource limits, we can test the pattern matching speed with a large number of synthesized pattern vectors. Table 1 has shown the matching speed of different pattern library sizes and embedding vector dimension combinations, from which we can see that even with the pattern library enlarged to 10000 stored patterns with dimension 512, the query and matching can still be finished in 0.4s. The time overhead is fast enough to be ignored in the whole process. Figure 8 shows the time comparison for mask convergence speed with or without pattern, 10 cases were tested after 800 patterns were inserted into the pattern library. Iterations of mask updates required can be significantly reduced by 93.6% on average.

At last, we need to verify the overall performance of our AdaOPC framework by comparing it with two baseline methods that we used in different branches for critical/non-critical patterns. After inserting 800 patterns with optimized masks into the library, we randomly tested 10 patterns, and Table 2 shows that AdaOPC can achieve comparable EPE/PVBand performance with ILT-GPU approach with 10× acceleration and no accuracy loss. On the other hand, when compared with DAMO-DMG with 1 round lithography verification, AdaOPC shows much better performance with comparable speed. Notice that in our case, the embedding vector dimension is 256. As shown in Table 1, even if the library gets large to 10000 patterns, the matching time is still around 0.2s, which is marginal to the complete OPC process time cost shown in Table 2. To summarize, the AdaOPC framework achieves performance-speed dual optimization.

7 CONCLUSION

In this paper, we proposed a self-adaptive OPC framework for mask optimization for real designs by inspecting the characteristics of a full design. We proposed an extensible OPC solver selector to choose an appropriate solver for patterns with different complexity. Additionally, we also built a dynamic pattern library to reuse optimized masks for repeating patterns with the same geometric shape. We use supervised contrastive learning to embed patterns into vectors and propose a graph-based search strategy for fast pattern matching. At last, we validate the mask reusability by proving pattern shift equivariance property and proposed a practical shift calibration tool. Extensive experiments have shown our frame can achieve OPC speed-robustness co-optimization for real design patterns.

ACKNOWLEDGMENTS

This work is supported The Research Grants Council of Hong Kong SAR (Project No. CUHK14208021).

REFERENCES

[1] D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nanolithography," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1453–1472, 2013.

[2] J.-S. Park, C.-H. Park, S.-U. Rhie, Y.-H. Kim, M.-H. Yoo, J.-T. Kong, H.-W. Kim, and S.-I. Yoo, "An efficient rule-based opc approach using a drc tool for 0.18/spl mu/m asic," in *Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525)*. IEEE, 2000, pp. 81–85.

[3] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 1591–1594.

[4] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 8, pp. 1345–1357, 2016.

[5] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in *Optical Microlithography XXVIII*, vol. 9426. International Society for Optics and Photonics, 2015, p. 94260X.

[6] A. Poonawala and P. Milanfar, "Mask design for optical microlithography—an inverse imaging problem," *IEEE Transactions on Image Processing*, vol. 16, no. 3, pp. 774–788, 2007.

[7] Y. Ma, W. Zhong, S. Hu, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5069–5082, 2020.

[8] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A gpu-enabled level set method for mask optimization," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1835–1838.

[9] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "Gan-opc: Mask optimization with lithography-guided generative adversarial nets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2822–2834, 2019.

[10] B. Jiang, H. Zhang, J. Yang, and E. F. Young, "A fast machine learning-based mask printability predictor for opc acceleration," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 412–419.

[11] H. Geng, W. Zhong, H. Yang, Y. Ma, J. Mitra, and B. Yu, "Sraf insertion via supervised dictionary learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2849–2859, 2019.

[12] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "Damo: Deep agile mask optimization for full chip scale," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[13] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "Develset: Deep neural level set for instant mask optimization," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[14] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, and E. F. Young, "Neural-ilt: Migrating ilt to neural networks for mask printability and complexity co-optimization," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[15] H. H. Hopkins, "The concept of partial coherence in optics," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 208, no. 1093, pp. 263–277, 1951.

[16] N. B. Cobb, *Fast optical and process proximity correction algorithms for integrated circuit manufacturing*. University of California, Berkeley, 1998.

[17] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "Mosaic: Mask optimizing solution with process window aware inverse correction," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[18] Y. Watanabe, T. Kimura, T. Matsunawa, and S. Nojima, "Accurate lithography simulation model based on convolutional neural networks," in *Optical Microlithography XXX*, vol. 10147. International Society for Optics and Photonics, 2017, p. 101470K.

[19] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "Lithogan: End-to-end lithography modeling with generative adversarial networks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[20] H.-C. Shao, C.-Y. Peng, J.-R. Wu, C.-W. Lin, S.-Y. Fang, P.-Y. Tsai, and Y.-H. Liu, "From ic layout to die photograph: A cnn-based data-driven approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 5, pp. 957–970, 2020.

[21] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[24] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[25] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3733–3742.

[26] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *arXiv preprint arXiv:1808.06670*, 2018.

[27] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[28] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 661–18 673, 2020.

[29] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[30] S. Banerjee, Z. Li, and S. R. Nassif, "Iccad-2013 cad contest in mask optimization and benchmark suite," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 271–274.

[31] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.