AdaOPC 2.0: Enhanced Adaptive Mask Optimization Framework for via Layers

Wenqian Zhao, Xufeng Yao, Shuo Yin, Yang Bai, Ziyang Yu, Yuzhe Ma, *Member, IEEE*, Bei Yu, *Senior Member, IEEE*, and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—Optical proximity correction (OPC) is a widely used technique to enhance the printability of designs in various foundaries. Recently, there has been a growing interest in using rigorous numerical optimization and machine learning to improve the robustness and efficiency of OPC. Our research focuses on developing a self-adaptive OPC framework that leverages the properties of pattern distribution and repetition in design layouts to optimize the correction process. We observe that different subregions in a design layer have varying pattern complexities, and many patterns repeat themselves throughout the layout. By exploiting these properties, we propose a framework that adaptively selects the most suitable OPC solvers from an extensible pool to optimize the correction process for each pattern based on its complexity. This approach allows for a co-optimization of speed and accuracy. Additionally, we introduce a graph-based dynamic pattern library that reuses optimized masks for repeated patterns, further accelerating the OPC flow. Our experimental results demonstrate a significant improvement in both performance and efficiency using our proposed framework.**

***Index Terms*—Allocation, design for manufacturability, design reuse, layout, mask optimization, optical proximity correction (OPC).**

I. Introduction

THE CONTINUOUS shrinking of VLSI technology nodes has led to a significant impact on the manufacturability of integrated circuits. This is due to the non-negligible lithography proximity effect [1], which can cause issues during the printing process. Resolution enhancement techniques (RETs) are employed to address this challenge and improve the printability of the lithography process. One of the most widely used RETs is optical proximity correction (OPC), which optimizes mask printability by compensating for the diffraction effect that occurs during the lithography process.

OPC approaches can be categorized into: 1) rule-based OPC [2]; 2) model-based OPC [3], [4], [5]; 3) inverse lithography technique (ILT)-based OPC [6], [7], [8]; and 4) machine learning (ML)-based OPC [9], [10], [11], [12]. Rule-based methods provide a heuristic solution to the problem, offering a simple and fast approach, but they are only appropriate for less aggressive designs. On the other hand, model-based OPCs employ mathematical modeling of the lithography process to adjust the edge fractures of the mask accurately, ensuring high fidelity. However, these methods are limited by the solution space when dealing with advanced technology nodes. ILT-based methods, on the other hand, address the OPC problem by solving the inverse problem of the imaging system using an optimized objective function. This approach represents the most effective analytical method for tackling the OPC problem. With the rapid advancement of ML algorithms and hardware in recent years, ML-based OPCs have demonstrated impressive acceleration in OPC workflows and have gained prominence in the academic field of design for manufacturing (DFM) [13], [14], [15]. Studies, such as [9], have employed deep learning models for initial mask generation to reduce the number of iterations required for ILT. Additionally, Jiang et al. [16] utilized a deep learning model to simulate the conventional ILT correction process. However, it is important to note that a significant drawback of ML models is their opaque nature, as they are driven by data and lack interpretability. Such methods do not guarantee effectiveness for critical patterns. In conclusion, no approach is flawless or universally superior to others. Patterns with varying complexities necessitate different approaches.

To achieve efficient and desired OPC results on real designs, it is necessary to conduct a systematic analysis of pattern distribution and complexity. Through careful examination of a real design, we have identified several properties that can be utilized to aid in this analysis. One such property is the presence of varying pattern densities, indicating that certain regions exhibit high density while others are more sparse. This *diversity*, as depicted in Fig. 1, suggests the need for different types of OPC solutions in different regions. Furthermore, upon conducting a more detailed examination of each subregion, we have observed a certain degree of similarity in the pattern distribution across different subregions. Numerous patterns are recurrently positioned throughout the entire design layer, featuring comparable geometric shapes but varying locations. This pattern repetition allows us to exploit their shared geometric characteristics, leading to the idea that the OPC solution for one pattern can be applied to similar patterns, thereby enhancing efficiency. Inspired by these findings, we introduce a self-adaptive framework called AdaOPC, specifically designed for performing OPC on real designs.

Manuscript received 19 September 2023; revised 14 February 2024; accepted 7 March 2024. Date of publication 18 March 2024; date of current version 22 August 2024. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Project CUHK14208021. This article was recommended by Associate Editor P. A. Beerel. *(Corresponding author: Wenqian Zhao.)*

Wenqian Zhao, Xufeng Yao, Shuo Yin, Yang Bai, Ziyang Yu, Bei Yu, and Martin D. F. Wong are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: wqzhao@cse.cuhk.edu.hk; prov@hkbu.edu.hk).

Yuzhe Ma is with the Microelectronics Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou 510530, China.

Digital Object Identifier 10.1109/TCAD.2024.3378600

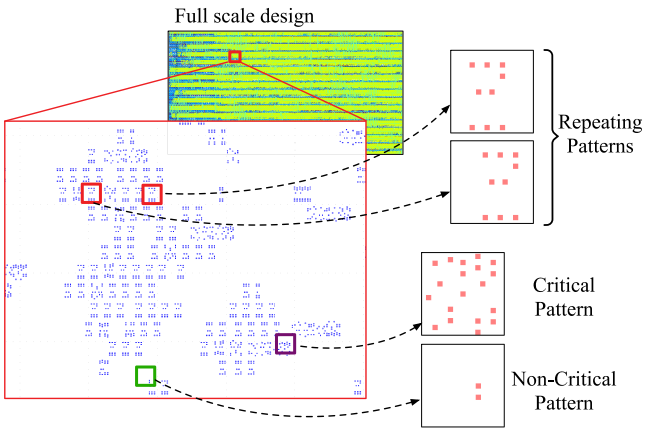

Fig. 1. Visualization of a real design layer. Two key observations served as the motivation for our OPC framework design. First, patterns were found to be distributed unevenly throughout the design layout, exhibiting varying levels of complexity. We classified intricate patterns as critical, while simple patterns were labeled as noncritical. Second, a significant proportion of the patterns displayed a high degree of repetition across the entire layout.

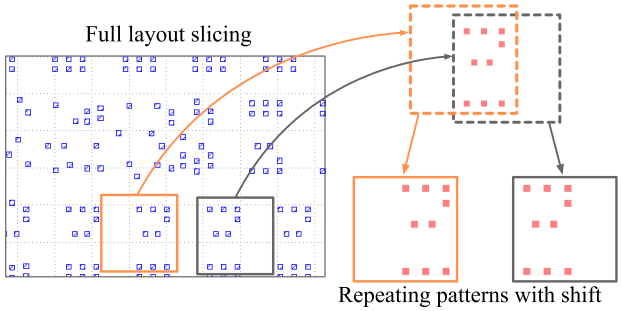

Fig. 2. Slicing repeating full layout inevitably causes some location shift on repeating patterns.

First, AdaOPC incorporates pattern analysis capabilities, enabling the classification of subregions as either critical or noncritical. This classification allows for the appropriate selection of the OPC solver. In particular, densely scattered subregions exhibit not only the diffraction effect but also the optical interference caused by neighboring components, both of which collectively impact the final printed image. These intricate patterns, requiring robust and rigorous numerical optimization methods, are considered critical and are better suited for achieving higher manufacturability. On the other hand, subregions with sparsely scattered patterns are simpler, making them more amenable to mask optimization processes utilizing ML models, which offer superior inference speed through learned representations.

Second, considering the presence of numerous recurring patterns on the design layer, all sharing the same geometric shape as depicted in Fig. 1, we explore the possibility of reusing optimized masks for these repetitive patterns to eliminate redundant OPC iterations. However, our idea faces three significant obstacles.

1) Slicing the large design layout into smaller patterns, as illustrated in Fig. 2, unavoidably introduces a shift in the location of patterns with identical geometric shapes. The question arises: Can an optimized mask with a location shift be effectively reused? And if so, how?
2) How can we accurately and efficiently match a given query pattern with the corresponding pattern from a vast repository of stored patterns?
3) How to measure the geometric similarity of patterns with location shift?

To address the aforementioned questions, we have developed a *dynamic pattern library* with online updating capabilities, enabling us to store and reuse both repeating patterns and optimized masks. This is achieved through the construction of a dynamic hierarchical graph. Furthermore, we have mathematically demonstrated the shift equivariance property of the lithography process, affirming the feasibility of mask reuse. By accurately calculating the shift of the design pattern and calibrating the mask accordingly, we ensure the effective reuse of masks despite pattern location shifts. We have implemented a graph-based approximation method for efficient pattern matching, enabling us to quickly identify the nearest neighbors within a short query time. We summarize the contributions of this article as follows.

1) We propose a self-adaptive workflow that allows for flexible selection of OPC solvers.
2) We prove the feasibility of mask reuse to speed up the OPC process for real design patterns and provide an efficient mask shift calibration method in practice.
3) We generate design patterns embedding by supervised contrastive learning for similarity measurement and pattern matching.
4) We construct a dynamic pattern library using a hierarchical graph with online update along with a greedy graph-based nearest neighbor search (NNS) for fast matching.
5) We bring a new weighting strategy during last modification stage to handle the sizing problem.
6) 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.

II. Preliminaries

A. Lithography Simulation Model

In the lithography process, an input mask $\mathbf{M} \in \mathbb{R}^{h\times w}$ is projected onto a wafer plane using layers of optical lenses. The resulting aerial image, denoted by $\mathbf{I} \in \mathbb{R}^{h\times w}$, is then used to create a coating on the wafer using photoresist, which ultimately forms the final pattern $\mathbf{Z} \in \mathbb{R}^{h\times w}$. The conventional approach to simulating the lithography process involves two sequential components: the optical projection model and the photoresist model.

The Hopkins diffraction model [17] has been extensively utilized for the mathematical analysis of coherent imaging systems during the projection process. However, an alternative approach based on singular value decomposition (SVD) has gained popularity due to its computational complexity.

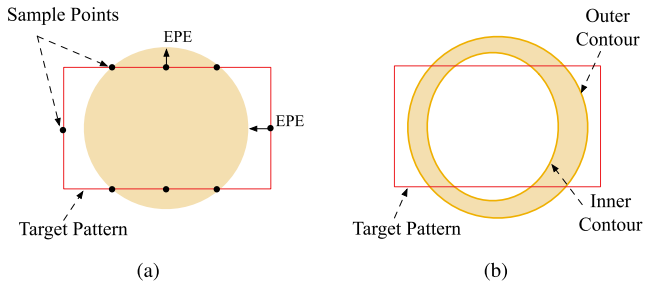

Fig. 3. OPC evaluation criteria. (a) Visualization of EPE measurement. (b) Visualization of PVBand.

This SVD-based approximation, initially proposed by [18], decomposes the Hopkins diffraction model into a summation of coherent systems through eigenvalue decomposition

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

where k denotes the kernel number in the coherent system. h_k is the kernel parameters itself. w_k is the corresponding weight assigned to the kth kernel. A Kth-order approximation is illustrated in [19] as

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

In our experiment, we selected $K = 24$ for the approximation. Following the optical simulation, the resulting lithography intensity $\mathbf{I}$ is fed into the photoresist model, which generates the final binary pattern $\mathbf{Z}$ using an exposure resist threshold I_{th}

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

Various ML-based approaches have been proposed for lithography simulation. For instance, Watanabe et al. [20] employed a CNN to determine the function model for resist model simulation. In another study, Ye et al. [21] introduced LithoGAN, a GAN-based framework designed to map input masks to output resist patterns. Additionally, Shao et al. [22] proposed a two-stage DNN-based framework that addresses the mask-to-SEM prediction as a domain transfer problem, utilizing CycleGAN [23] to learn the transfer process.

While DNN models often offer a speed advantage, we have utilized the Hopkins model due to its analyzability. By employing a white-box model, we gain the ability to mathematically analyze the pattern shift equivariance property during the lithography process.

B. OPC Evaluation Criteria

Edge Placement Error (EPE): Following the lithography process, the printed image on the wafer may exhibit geometric distortion compared to the intended design target. This distortion is commonly assessed using the EPE metric. The measurement of EPE is illustrated in Fig. 3(a), where a set of measuring points is sampled along the boundary of the target design pattern, encompassing both horizontal and vertical edges. For each location (x, y), if the distance $D(\cdot)$ between the printed image and the target exceeds a predefined threshold th_{EPE} at a specific measuring point, it is identified as an EPE violation

$$\text{EPE\_violation}(x,y) = \begin{cases} 1, & D(x,y) \geq th_{\text{EPE}} \\ 0, & D(x,y) < th_{\text{EPE}}. \end{cases} \quad (4)$$

Process Variation Band (PV Band): In practical lithography applications, process variation can introduce deviations in the final printed images, potentially resulting in printing failures. Printed images may exhibit diverse contour outcomes depending on different lithography conditions, such as focus/defocus depth and incident light intensity. To assess the printing robustness, the PV Band is defined as the discrepant (XOR) region between the innermost and outermost contours, as depicted in Fig. 3(b). The PV Band serves as a measure to evaluate the impact of process variation on printing quality

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

where the size of the pattern is denoted by N. The printed pattern of the outer contour is represented by $\mathbf{Z}_{\text{out}}$, while the inner contour is represented by $\mathbf{Z}_{\text{in}}$.

III. Adaptive Framework

A. Workflow Overview

Our proposed workflow is visualized in Fig. 4. First, we introduce the OPC solver selection module in Section III-B, which chooses the appropriate OPC solver for different patterns. In Section IV-B, we use supervised contrastive learning to discuss how patterns are embedded into high-dimensional vectors for pattern matching in the library. Finally, Section V discusses mask reusability and requirements, proving shift equivariance during lithography to claim the practicality of the approach. Additionally, a solution using shift calibration is provided.

B. OPC Solver Selection

To accommodate the varying complexity of different patterns, our framework incorporates a flexible solver pool that selects appropriate OPC solutions. We categorize the sliced design patterns into two types: critical and noncritical patterns. A solver selector module is employed to determine the OPC solver to be used. This solver selector can be viewed as a 2-class classifier constructed using a simple deep-learning classification model. As the backbone network, we utilize ResNet-18 [24] and train it with the objective of minimizing the cross-entropy loss L

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

In the training process, each sample i is associated with a binary label y_i, indicating whether it belongs to the critical pattern class (1) or not (0). The classifier model predicts the probability p_i for each sample. The objective of training is to

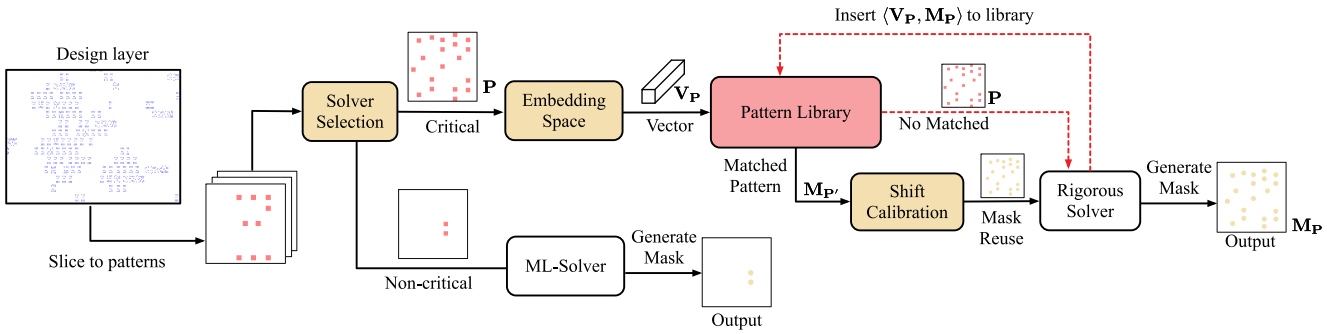

Fig. 4. Overall workflow of AdaOPC is depicted using colored blocks to represent functional modules. The red dashed lines indicate the flow of library updates within the workflow.

minimize the discrepancy between the predicted probabilities p_i and the corresponding labels y_i. Despite its simplicity, this straightforward combination of a simple network architecture and the associated loss function yields fast and accurate predictions for pattern classification without any additional complexities or embellishments.

For the ML-Solver handling noncritical patterns, we adopt a generative neural network model inspired by DAMO-DMG [12], which is the previous state-of-the-art (SOTA) OPC solver for via layer. Specifically, we utilize U-Net++ with residual connection as the underlying model structure. We follow a similar strategy to train this generative model as outlined in [12]. However, there is a distinction in our training data. Instead of relying on a DNN simulator as in [12], we curate our own training dataset comprising patterns sourced from a real full-scale design. Additionally, the masks in our dataset are generated using a robust OPC engine that employs an authentic lithography model. This data preparation approach aligns more closely with the real-world OPC scenario, where the lithography model serves as the sole source of ground truth information.

When handling critical patterns, we employ a rigorous optimization method described in [19], utilizing GPU acceleration through CUDA and fully optimized memory control. This choice is made even though deep learning approaches have achieved impressive performance in holistic evaluations on certain pattern test sets. While data-driven black-box deep learning models excel at mimicking and reversing diffraction effects, they may encounter challenges when confronted with optical interference resulting from complex neighboring components. In such scenarios, the rigorous numerical solver offers an analytical solution that remains unaffected by the geometric complexity of patterns. Furthermore, in an actual OPC situation involving a new design and potentially a new lithography engine, patterns, and optimized masks generated using robust methods can serve as a dataset for training the ML model to adapt to these new settings.

It is important to note that the solver pool is designed to be extensible. This means that any OPC solution with specific strengths for particular patterns has the potential to be incorporated as a replacement or complementary candidate within the pool. If there are more than two solvers in the pool, the classifier loss can be easily modified as follows:

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

where C represents the number of pattern classes, which is equal to the number of corresponding OPC solvers. The label y_{ic} denotes whether a pattern belongs to category c (1) or not (0). By employing this approach, we can effectively convert the problem into a multiclassification scenario, allowing for the inclusion of multiple OPC solvers in the solver pool.

C. Empirical Risk Minimization for Selector

As mentioned previously, patterns from the same design may lead to insufficient training set to train the selection model with robustness, given that the original training is purely supervised and the label is binary (one-hot encoded for multiple solvers). The simple labeling may lead to overfitting on the training patterns, which even have similar patterns to the evaluation set. This seems inevitable because these patterns are repetitive. However, we cannot expand with more designs, so some augmentation techniques are needed.

For all the pattern and corresponding critical label distribution $(x, y) \sim P$. We have the objective to minimize the *expected empirical risk* of the trained model f

$$R(f) = \frac{1}{n}\sum_{1}^{n} l(f(x_i), y_i). \tag{8}$$

Note that here the sample $(x_1, y_1), \ldots, (x_n, y_n)$ is an approximation to the real distribution. A Gaussian noise on the distribution can help to reduce this risk term with additive Gaussian noise on both input pattern x and critical label y

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j \end{aligned} \tag{9}$$

where λ is chosen from a Beta distribution with value range $\lambda \in [0, 1]$. Overall, such augmentation can enhance the dataset distribution.

This method provides valuable assistance to the pattern selector in accurately discerning patterns with densities that are close to either critical or noncritical ones. This capability allows the selector to make well-informed decisions based on

the specific pattern density. This is an critical motivation in the workflow of AdaOPC. Not only is our motivation focus on time efficiency, the robustness is enhanced at this very first stage. By accurately detecting ambiguous areas, the risk of erroneous identification of critical patterns is mitigated, resulting in further optimization of overall robustness.

IV. Dynamic Pattern Library

Once we have filtered out simpler cases, employing a rigorous solver for critical patterns becomes necessary. To further enhance efficiency, we establish a dynamic pattern library to store pairs of patterns: the sliced pattern $\mathbf{P} \in \mathbb{R}^{h \times w}$ and its post-OPC mask $\mathbf{M_P} \in \mathbb{R}^{h \times w}$. This allows us to reuse masks from repeating patterns, thereby avoiding the redundant, time-consuming process of OPC iterations from scratch. The fundamental concept is to identify stored repeating patterns before performing OPC. An online update mechanism ensures that new patterns and their corresponding masks can be inserted into the library.

We utilize a graph structure inspired by [25] to construct the pattern library. In this structure, each node within the graph represents a stored pattern, and the connections between nodes indicate a high level of similarity between the corresponding patterns. Given the substantial number of patterns in a full design layout, the graph can reach a significant size. Conducting a naive search for the shortest distance between nodes through pairwise distance comparisons would prove impractical. To address this, we implement the following enhancements to improve matching efficiency.

1) *Sparse Neighborhood Graph Structure:* Nodes that are distant from each other exhibit sparse connections, reducing the overall number of edges in the graph.
2) *Graph Hierarchy:* We divide the graph into hierarchical layers, and each layer restricts the degree of nodes. Lower layers contain more edges, enabling a greedy search for nearest neighbors at each layer.

Please refer to Fig. 5 for visually representing of the hierarchical sparse graph structure.

A. Pattern Matching and Online Update

The objective of identifying the pattern with the closest geometric shape can be seen as an NNS problem. Drawing inspiration from [25], we employ the Hierarchical Navigate Small World (HNSW) algorithm to ensure efficient matching. The matching process, as depicted in Fig. 5, follows a greedy approach, traversing the graph from higher layers to the bottom layer. During this top-down traversal, a list of potential pattern nodes representing the nearest candidates is maintained. The list is updated whenever a closer pattern is encountered during the search, surpassing the distance of one of the existing candidates. This matching strategy is based on the concept of proximity graph NNS. For a detailed understanding of the pattern-matching search strategy at each hierarchical layer, refer to Algorithm 1.

Once we reach the bottom layer, patterns in the candidate list C that have a distance smaller than the threshold σ are considered matches. If the smallest distance in C is still larger than σ, we classify it as a new pattern. This approach maintains its speed and accuracy even as the graph expands with continuous insertions of new patterns into the library.

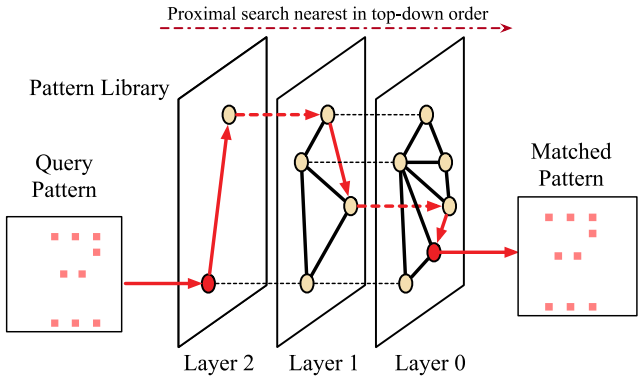

Fig. 5. Graph-based pattern matching flow is visually represented, illustrating the traversal of the query design pattern $\mathbf{P}$ greedily traversing the hierarchical graph. The nearest node reached at layer 0 corresponds to a match pattern $\mathbf{P}'$, which has the most similar geometric shape with $\mathbf{P}$.

Algorithm 1 Graph-Based Pattern Matching Greedy Search

Input: Query pattern $\mathbf{P}$, starting nodes q_s, number of nearest neighbor to return k,layer number l, distance measurement $d(\cdot)$.
Output: Nearest pattern candidates C.

$V \leftarrow q_s$; ▷ Visited nodes
$W \leftarrow q_s$; ▷ Waiting list of nodes to visit
$C \leftarrow q_s$;
while $|W| > 0$ **do**
  $q^* \leftarrow$ nearest pattern from W to $\mathbf{P}$;
  $q_f \leftarrow$ furthest pattern from C to $\mathbf{P}$;
  if $d(\mathbf{P}, q^*) > d(\mathbf{P}, q_f)$ **then**
    break;
  end if
  for $e \in$ `neighbor`(q^*) in layer l **do**
    if $e \notin V$ **then**
      $V \leftarrow V \cup \{e\}$;
      $q_f \leftarrow$ furtherest pattern from C to $\mathbf{P}$;
      if $d(\mathbf{P}, e) < d(\mathbf{P}, q_f)$ or $|C| < k$ **then**
        $W \leftarrow W \cup \{e\}$;
        $C \leftarrow C \cup \{e\}$;
        if $|C| > k$ **then**
          Remove furthest pattern from C to $\mathbf{P}$;
        end if
      end if
    end if
  end for
end while

The pattern library follows an online update method. When encountering a new pattern that has no matches, the mask undergoes OPC iterations starting from scratch for optimization. Subsequently, the library inserts the pattern and its optimized mask as a new node, updating the edge hierarchy of the graph to accommodate the new pattern. The detailed steps of the online update process are outlined in Algorithm 2.

According to Algorithm 2, the new pattern is inserted into one of the hierarchical layers with a decaying probability. At the same layer, edges are added between this pattern and the top k nearest patterns. As the neighboring nodes' degrees increase, an edge reconnection is performed when the degree exceeds the upper bound k. Consequently, the degree of each

Algorithm 2 New Pattern Insertion and Graph Update

Input: hierarchical graph G, new pattern $\mathbf{P}$, total layer number L, G's starting nodes q_s, max degree M.
Output: updated hierarchical graph G.

$l \leftarrow$ `random`$(0, L)$; ▷ exponentially decaying probability
for $l_c \leftarrow L, ..l$ **do**
  $C \leftarrow$ `search`(P, q_s, k, l_c); ▷ Algorithm 1
  $q_s \leftarrow$ nearest pattern of q in C;
end for
for $l_c \leftarrow l, ..0$ **do**
  Insert $\mathbf{P}$ to layer l_c of G; ▷ add $\mathbf{P}$ into graph
  C $\leftarrow$ `search`(P, q_s, k, l_c); ▷ Algorithm 1
  `neighbors`$((P)) \leftarrow$ top M nearest patterns in C;
  for $e \leftarrow$ `neighbors`$(\mathbf{P})$ **do**
    Add edge (P, e);
    if Degree of $e > M$ **then**
      `neighbors`$(e) \leftarrow$ top k nearest patterns to e;
      Remove all edges connecting e;
      Create edges e with each one in `neighbors`(e);
    end if
  end for
end for

node in the graph is limited to k. It is worth noting that the number of edges directly affects the complexity of the matching process. The utilization of a sparse hierarchical graph facilitates efficient searching as the graph size increases.

To assess the similarity of vectors, we have proposed several distance metrics. One such metric involves using the inner product of two vectors to evaluate the difference in direction between them

$$\text{Inner}\left(\mathbf{V}_{\mathbf{P}_1}, \mathbf{V}_{\mathbf{P}_2}\right) = \mathbf{V}_{\mathbf{P}_1} \cdot \mathbf{V}_{\mathbf{P}_2} = \sum_{i=0}^{k} V_{\mathbf{P}_1,i} V_{\mathbf{P}_2,i}. \tag{10}$$

In the equation, $\mathbf{V}_{\mathbf{P}_1}$ and $\mathbf{V}_{\mathbf{P}_2}$ represent the embedded vectors of patterns $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively. $V_{\mathbf{P}_1,i}$ denotes the ith element of vector $\mathbf{V}_{\mathbf{P}_1}$. The dimension of the embedded vector is set to $k = 256$. It is important to note that the inner product used in the equation violates the positivity property, where an element can be closer to some other element than to itself.

Unlike inner product, *cosine similarity* does not violate the positivity property and can therefore be used to measure the similarity between two vectors in an inner product space

$$\begin{aligned} d_{\text{Cosine}}\left(\mathbf{V}_{\mathbf{P}_1}, \mathbf{V}_{\mathbf{P}_2}\right) &= 1.0 - \frac{\mathbf{V}_{\mathbf{P}_1} \cdot \mathbf{V}_{\mathbf{P}_2}}{\left\|\mathbf{V}_{\mathbf{P}_1}\right\| \left\|\mathbf{V}_{\mathbf{P}_2}\right\|} \\ &= 1.0 - \frac{\sum_{i=0}^{k} V_{\mathbf{P}_1,i} V_{\mathbf{P}_2,i}}{\sqrt{\sum_{i=0}^{k} {V_{\mathbf{P}_1,i}}^2} \sqrt{\sum_{i=0}^{k} {V_{\mathbf{P}_2,i}}^2}}. \end{aligned} \tag{11}$$

An alternative approach is to use *Euclidean distance*, where the embedding metric space is treated as a Euclidean space. Each vector represents a position in Cartesian coordinates, and the similarity between two vectors can be determined by calculating the squared -2 norm of the difference between the coordinates

$$d_{\text{Euclid}}\left(\mathbf{V}_{\mathbf{P}_1}, \mathbf{V}_{\mathbf{P}_2}\right) = \left\|\mathbf{V}_{\mathbf{P}_1} - \mathbf{V}_{\mathbf{P}_2}\right\|_2^2 = \sqrt{\sum_{i=0}^{k} \left(V_{\mathbf{P}_1,i} - V_{\mathbf{P}_2,i}\right)^2}. \tag{12}$$

Any metric that follows the principles of NNS can be used as a feasible similarity measurement metric, allowing for the exploration of different metrics based on various embedding spaces. For our implementation, the embedding space learned with the pattern feature loss mentioned in (7), the optimal pattern matching accuracy is achieved by Euclidean distance measurement. Here, we emphasize the motivation difference between the selection and library stages, although they both involve a pattern embedding process. Therefore, these two stages shall use different metrics for their embedding objectives. The cross-entropy loss in (7) helps the classification of all patterns into groups based on explicit pattern density. On the other hand, the library stage does not put any explicit label on each pattern but requires a hidden embedding space where the similarity of all pattern pairs can be quantified and compared. We have compared several metrics for pattern-matching in our implementation and found that Euclidean distance measurement (12) performs best matching accuracy.

B. Embedding Space Construction

In order to utilize a stored mask from the library, it is necessary to match a pattern with the same geometric shape. However, directly comparing the geometric similarity of two patterns is a complex task. To address this challenge, we have developed an embedding metric space that captures the geometric properties through a high-dimensional vector representation, denoted as $\mathbf{VP}$. Instead of storing the original $\langle \mathbf{P}, \mathbf{MP} \rangle$ pair in the library, it is replaced with the $\langle \mathbf{VP}, \mathbf{MP} \rangle$ pair. This enables the determination of whether two patterns are the same by employing a similarity metric to compare the embedded vectors.

The construction of the embedding space is achieved by converting it into a feature extraction process using a deep learning model. The metric space is then established through deep metric learning, with the embedded vector being the output of an embedding neural network. The deep learning model for the embedding process consists of two modules.

1) The *Encoder*, denoted as $Enc(\cdot)$, is responsible for encoding each input pattern $\mathbf{P}$ into a feature map $\mathbf{F}_P \in \mathbb{R}^{h \times w \times c}$. The feature map has spatial dimensions of h and w, and c represents the number of channels.
2) The *Projector*, denoted as $Proj(\cdot)$, takes the feature map $\mathbf{F}_P$ as input and embeds it into a representation vector $\mathbf{VP} \in \mathbb{R}^k$. During the training stage, the output $Proj(\mathbf{F}_P)$ is normalized to lie on the unit hypersphere in $\mathbb{R}^k$ for loss calculation.

Therefore, the embedding process is formulated as

$$\mathbf{V}_\mathbf{P} = \text{Proj}(\text{Enc}(\mathbf{P})) \in \mathbb{R}^k. \tag{13}$$

In previous deep-learning-based OPC approaches [9], [12], [21], UNet or its variant UNet++ was commonly chosen as the backbone structure. However, the OPC problem imposes a strict requirement where the output mask must maintain the exact resolution as the input design. This limitation significantly narrows down the options for selecting network backbone structures.

We gain flexibility in choosing various network structure candidates by adopting an embedding process without such

limitations. Our approach intentionally selects one of the most widely used structures, ResNet-18 [24], as the encoder. The input pattern $\mathbf{P}$ is a 2-D image with dimensions of 2048 × 2048. To mitigate heavy computational load and reduce time delay, we employ a greedy downsampling strategy to reduce the pattern size to 256×256 before feeding it into the neural network. This downsampling does not noticeably degrade performance.

Following the original ResNet-18 structure, we add a depthwise convolution layer to reduce the feature channel size from 512 to 256. At the end of the neural network, a linear layer is applied to transform the resulting 3-D feature into the final 1-D embedded vector **VP**. The size of **VP** is a tradeoff, where a larger size indicates higher matching accuracy but slower similarity computation and matching speed. Through experimentation, we have determined that a size of 256 strikes a balance between good performance and negligible matching time with experiments.

The embedding space $\mathbf{S}$ is specially designed with certain objectives.

1) Patterns with the same shape exhibit similar embedded vectors with the shortest distance between them.
2) Patterns with different shapes are sparsely clustered in the embedding space and located far apart.

To train such an embedding space, a sufficient amount of data is required, consisting of both different patterns and instances of the same pattern. During training, data from the same pattern is treated as positive samples, and the embedded vectors are encouraged to be as close as possible, indicating higher similarity. On the other hand, embedded vectors of different patterns are treated as negative samples and are pushed as far away from each other as possible. To create the training dataset, we extract numerous patches of patterns from a real full-scale design. This dataset provides diverse pattern samples for training the embedding space.

Data Preparation: The cropping process consists of two steps to generate the necessary positive and negative samples for training. In the first step, we randomly select anchor points along the design layer. In the second step, we apply random shifts around each anchor point to obtain a certain number of patches. These patches have the same pattern within a square patch but varying relative positions. Labeling the patterns based on their anchor point ensures that each batch of training data adheres to the requirement of positive and negative samples.

Supervised Contrastive Loss: When training the neural network to learn how to embed patterns into representative vectors, the traditional cross-entropy loss may not adequately capture interclass distances or handle noisy labels. In the domain of self-supervised learning, there exists a family of losses based on metric distance learning [26], [27], [28], [29], among which contrastive loss [30] is particularly powerful for learning representative embeddings. Inspired by the work of [31], we extend the contrastive loss to a supervised contrastive loss. This is achieved by genuinely generating and labeling all positive and negative samples during the data preparation stage. It ensures that the labels accurately reflect the nature of the samples. As mentioned, the representation vector $\mathbf{z}$ is obtained by normalizing the embedded vector $\mathbf{V_P}$

$$\mathbf{z} = \text{normalize}(\text{Proj}(\text{Enc}(\mathbf{P}))) \in \mathbb{R}^k. \tag{14}$$

Then, the loss function is formulated as

$$\mathcal{L}_{\text{supCon}} = -\sum_{i\in I}\frac{1}{|J(i)|}\sum_{j\in J(i)}\log\frac{\exp(\mathbf{z}_i\cdot\mathbf{z}_j/\tau)}{\sum_{a\in A(i)}\exp(\mathbf{z}_i\cdot\mathbf{z}_a/\tau)}. \tag{15}$$

In the context of the training batch, let i be the anchor index and j be the anchor index of the positive samples. The set $A(i) = I\backslash i$ represents all the anchor indices in this batch except for i. Consequently, $A(i)\backslash J(i)$ represents the anchor indices of the negative samples. Here, τ is a scalar temperature parameter. The term $\exp(\mathbf{z}i\cdot\mathbf{z}j/\tau)$ in the numerator represents the similarity between the positive sample pairs $\mathbf{z}i$ and $\mathbf{z}j$. On the other hand, the term $\exp(\mathbf{z}i\cdot\mathbf{z}a/\tau)$ in the denominator represents the similarity between all sample pairs, including the negative ones. By minimizing the loss, the training process aims to increase the similarity of positive samples and decrease the similarity of negative samples.

V. Mask Reuse With Shift Calibration

A. Mask Reusability

We operate under the assumption that if the query design pattern $\mathbf{P}$ matches a stored design pattern $\mathbf{P}'$ in the library with the same shape, repeating patterns can efficiently share masks for improved efficiency. However, as illustrated in Fig. 2, when the entire design is divided into smaller patterns, it becomes inevitable to encounter pattern location shifts $(\Delta x, \Delta y)$ between $\mathbf{P}$ and $\mathbf{P}'$. In real lithography and OPC flow, if no external factors influence the lithography process, the printed wafer image patch should not suffer from any geometric distortion. Instead, it will only exhibit an identical shift compared to the design pattern, as depicted in Fig. 6. Therefore, to reuse the mask, the first requirement is to ensure that the location shift during lithography does not result in any geometric distortions.

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 II-A as Litho$(\cdot)$ and the location shift as $\delta_{\Delta x,\Delta y}(\cdot)$, we show the following.

To demonstrate the feasibility of the mask shift calibration approach, we provide mathematical proof that the location shift remains unchanged before and after the lithography process. We represent the Hopkins diffraction model during lithography, as shown in Section II-A, as Litho$(\cdot)$. We denote the location shift as $\delta_{\Delta x,\Delta y}(\cdot)$. Through our mathematical analysis, we establish the following result.

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

$$\mathbf{P} = \text{Litho}(\mathbf{M_P}). \tag{16}$$

The following statement always holds:

$$\delta_{\Delta x,\Delta y}(\mathbf{P}) = \text{Litho}\big(\delta_{\Delta x,\Delta y}(\mathbf{M_P})\big). \tag{17}$$

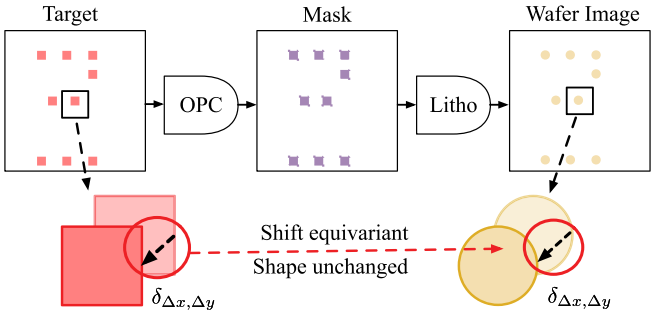

Fig. 6. Printed wafer image must exhibit an identical location shift to the design pattern without any geometric shape distortion.

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

$$\begin{aligned}
&\delta_{\Delta x,\Delta y}(\mathbf{P}(x,y)) = \mathbf{P}(x+\Delta x, y+\Delta y)\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x+\Delta x, y+\Delta y) \otimes \mathbf{M_P}(x+\Delta x, y+\Delta y) \right|^2\\
&= \sum_{k=1}^{N^2} w_k \left| \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\mathbf{M_P}\left(x+\Delta x+i-\frac{N}{2}, y+\Delta y+j-\frac{N}{2}\right)\right|^2\\
&= \sum_{k=1}^{N^2} w_k \left| \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\mathbf{M_P}\left(x+i-\frac{N}{2}+\Delta x, y+j-\frac{N}{2}+\Delta y\right)\right|^2\\
&= \sum_{k=1}^{N^2} w_k \left| \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\delta_{\Delta x,\Delta y}\left(\mathbf{M_P}(x+i-\frac{N}{2}, y+j-\frac{N}{2})\right)\right|^2\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x,y) \otimes \delta_{\Delta x,\Delta y}(\mathbf{M_P}(x,y))\right|^2\\
&= \mathrm{Litho}\left(\delta_{\Delta x,\Delta y}(\mathbf{M_P}(x,y))\right). \qquad (18)
\end{aligned}$$

Therefore, (17) is proved. ■

Given that mask shift during lithography only leads to a printing shift, it is possible for repeating patterns in a design to share OPC-optimized masks with a straightforward shift correction. To achieve this, we select the matched mask $\mathbf{MP}'$ from the pattern library and apply a correction of $(-\Delta x, -\Delta y)$ to obtain the initial mask $\mathbf{MP}$ for pattern $\mathbf{P}$.

B. Pattern Shift Calibration

To calculate the shift between patterns $\mathbf{P}$ and $\mathbf{P}'$, we utilize pixel-level similarity analysis. Specifically, we compute the pixel-wise *cross-correlation* between $\mathbf{P}$ and $\mathbf{P}'$. This cross-correlation measurement reflects of the similarity between individual pixels, and the pixel with the highest response value on the correlation map indicates the position shift of the center point (x_{ctr}, y_{ctr}). However, it is important to note that performing cross-correlation computation on two large 2-D patterns can be time consuming. The computation process of cross-correlation is equivalent to convolving $\mathbf{P}$ with the rotation of $\mathbf{P}'$ by 180°, denoted as Rotate(·)

$$\mathrm{CrossCorr}\left(\mathbf{P}, \mathbf{P}'\right) = \mathrm{Conv}\left(\mathbf{P}, \mathrm{Rotate}(\mathbf{P}')\right). \qquad (19)$$

To expedite the computation, we employ Convolution and leverage the efficiency of fast Fourier transform (FFT) [32]. By utilizing FFT, we can calculate the pattern shift using the following formula:

$$\begin{aligned}
x^*, y^* &= \underset{x,y}{\mathrm{argmax}} \ \mathit{Conv_FFT}\left(\mathbf{P}, \mathrm{Rotate}(\mathbf{P}')\right)\\
\Delta x &= x^* - x_{ctr}, \ \ \Delta y = y^* - y_{ctr}. \qquad (20)
\end{aligned}$$

And the initial mask is corrected with

$$\mathbf{M_P} = \delta_{-\Delta x,-\Delta y}(\mathbf{M_{P'}}). \qquad (21)$$

In practical applications, we validate the calibrated mask by inputting it into the lithography model. Additionally, we may iterate one or two more times using the ILT solver, if necessary, to account for any potential noise introduced during the shift calibration process. To ensure consistency, we employ the same pattern size as described in [9], which is 2048 × 2048. With our implementation, the shift calculation time is less than 0.25 s when executed on a CPU.

VI. Post-Calibration Enhancement

After matching and reusing the precollected mask, the original workflow of AdaOPC [33] directly sends the calibrated mask into the ILT flow for further optimization Fig. 4, which is a safe approach to guarantee the quality of output mask. On the other hand, the final result (EPE/PVBand) is upper-bounded by the original ILT approach. We have dug into the patterns after the original AdaOPC and proposed a post-stage for mask correction to improve the mask quality further.

Given that AdaOPC [33] initially targets via layers, which possess rather simple geometric shapes but complicated distribution. As we analyze printed aerial images as well as evaluation metrics at each iteration, we realize that the printed aerial images are usually "circle"-shaped with smooth boundaries. In many cases, after the lithography stage, the neighboring components strongly affect the printed circle size for each target. Conventional ILT approaches, either using pixel-based methods or levelset-based methods, mainly targeting on healing the boundary to conquer the disconnection or boundary merging problem, as visualized in Fig. 7. For metal layers, irregular object patterns and geometric shapes may occasionally encounter such problems during lithography. However, this sizing problem is also critical and needs further handling to fix. Our evaluation of different test cases demonstrates that this is the main cause for most EPE numbers.

As a post-calibration stage, we propose a "Halo-weighting" process after matching a precollected mask derived from the pure ILT stage. In this stage, we strive to force the object sizes to be balanced and close the original via a filtering step on the loss matrix during the update. We added a weighting filter $\mathbf{H} \in \mathbb{R}^{h\times w}$ when the matched mask was sent to the rigorous solver. The weighting filter follows the original design shape with the highest weight on the boundary and gradually decreases as distance far from the boundary, whose heatmap looks like a "Halo" that fades off the boundary of via in the original design pattern, as visualized in Fig. 8.

Such weight filter is derived with two consecutive steps: the first step is boundary derivation. We apply a morphological operation on the binary design image, calculating the

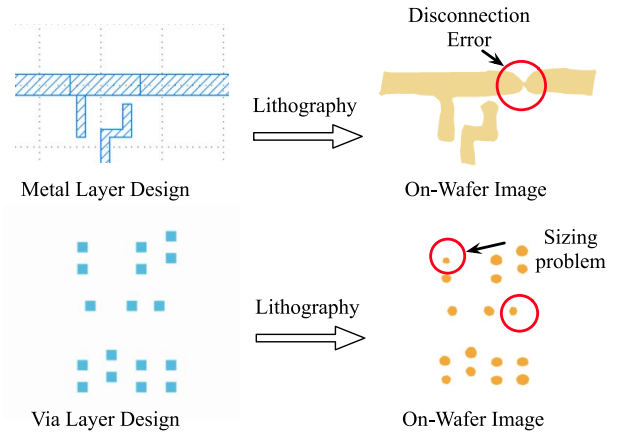

Fig. 7. Visualization of the sizing problem in Via layer mask optimization problem, in comparison with conventional metal layer. Some printed components are too small or too big, which may cause problem during manufacturing.

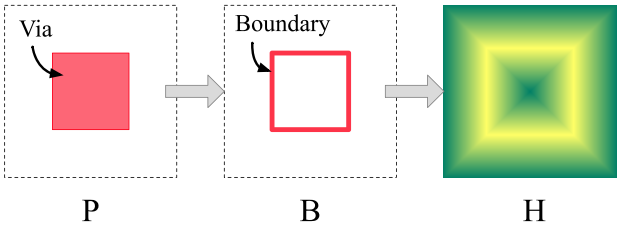

Fig. 8. Visualized derivation of halo weight filter $\mathbf{H}$. The square on the left is a patch $\mathbf{B}$ from original design. The square in the middle is the derived boundary $\mathbf{B}$ with morphological gradient from (22). The heapmap square on the right is the halo weight filter with higher value on the brighter area and lower value on the greener area ($0 \sim 1$).

Morphological Gradient to retrieve the boundary $\mathbf{B} \in \mathbb{R}^{h \times w}$ matrix of each pattern $\mathbf{P}$

$$\mathbf{B} = \mathbf{P} \oplus b - \mathbf{P} \ominus b \tag{22}$$

where the $\mathbf{P} \oplus b$ and $\mathbf{P} \ominus b$ denote the "dilation" and "erosion" operations. $\mathbf{b}$ is a grayscale structuring element used in computer graphics (here, the pattern is regarded as a 1-channel picture)

$$b(x) = \begin{cases} 0, & \text{if} \quad \|x\| \leq 1 \\ -\infty, & \text{otherwise.} \end{cases} \tag{23}$$

The second step is to blur the derived boundary $\mathbf{B}$ for a gradually decreasing weight. One off-the-shelf algorithm is distance transformation. The weight along the "halo" is determined by its distance from the boundary of each component derived from (22), where the value at (x, y) is the distance to the boundary

$$\mathbf{H}(x, y) = 1 - \text{Norm}(\text{distance}(x, y, \mathbf{B})). \tag{24}$$

It is guaranteed that the weight filter shows a gradual decrease as the position moves farther from the boundary. We normalize the distance to control the value range into [0, 1]. We subtract 1 with this value such that the highest value on the boundary and the value range of $\mathbf{H}$ is still [0,1]. The distance is calculated using simple Euclidean distance for faster calculation not only to attain a faster calculation but also to bring a smoother yet effective weight distribution

$$\text{distance}(x, y, \mathbf{B}) = \max_{(x_b, y_b) \in \mathbf{B}} (\|x - x_b\|, \|y - y_b\|). \tag{25}$$

In our implementation, we can keep the dilation and erosion size as 1, so the boundary length is 3 ($+1/-1$). The weight distribution is visualized in Fig. 8, which shows smooth weight distribution along the halo within and outside the boundary of the via component. The result of adding this filter on the loss matrix during ILT after calibration is that: The boundary restoration will restrict the area closer to the boundary. On the other hand, the loss accumulated far from the boundary will be less counted, given that this is post-matched mask. The huge errors are already handled before being inserted in the library and no longer exist at this stage.

VII. Experimental Results

A. Implementation Settings

Our framework is mainly developed using Python. The ML components of our framework are implemented using the PyTorch library. On the other hand, the lithography and ILT modules are built using C/C++ and utilize the CUDA 11.3 toolkit for optimized performance. To evaluate the performance and speed of our framework, we conducted experiments on a CentOS-7 system equipped with an Intel i7-5930K CPU running at 3.50 GHz, along with an Nvidia GTX Titan X GPU. For our experiments, we employed the publicly available lithography engine from the ICCAD 2013 CAD Contest [34], which includes a set of 24 optical kernels. The photoresist intensity threshold was set to 0.055. We adopted a lithography wavelength of 193 nm, with a defocus range spanning ± 25 nm and a dose range of $\pm 2\%$. To identify EPE violations, we set the EPE violation threshold (th_{EPE}) to 15 nm.

B. GPU Reimplementation

Although the original AdaOPC already chose GPU to implement the CUDA acceleration of the lithography process at each iteration, we can tell in Fig. 9(b) that this bottleneck still hinders the optimization efficiency. Compared to the original AdaOPC, we reimplement the ILT process to bridge the memory bound when conducting inference and gradient backpropagation. First, we keep all the intermediate tensors/matrices and kernel weights on the high bandwidth memory (HBM), namely, the GPU memory. Within each iteration of the update, the calculation of the matrix is done on GPU while the weight being used and matrices being updated are stationary on GPU memory. The unnecessary CPU-GPU data movement and synchronization are saved. Apart from that, our CUDA kernel is optimized at the thread level to decrease the cache miss rate for efficiency improvement. Overall, the optimization speed-up can reach $2.2\times$ acceleration for the same amount of ILT calculation.

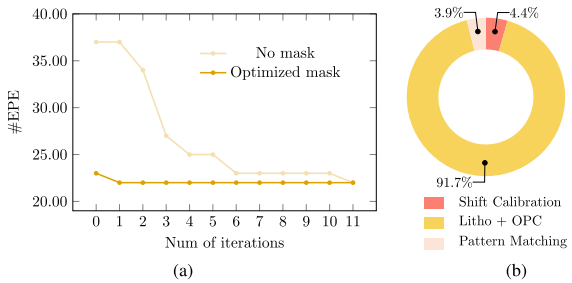

Fig. 9. (a) EPE convergence comparison. (b) Runtime breakdown of AdaOPC on critical patterns.

TABLE I
PATTERN-MATCHING SPEED ANALYSIS ON DIFFERENT EMBEDDING DIMENSION

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

C. Data Preparation

To ensure the authenticity of our OPC experiments, all data used in our study was derived from a real design extracted from a GDS file generated by the open-source layout generation tool OpenROAD [35]. We specifically sliced patterns of size 2048 × 2048, following the approach established in previous research works [8], [9], [12]. This is also the supported clip size of our Lithography engine. Therefore, we can avoid additional resize operations (upscale + downscale) that might not help pixel-based OPC task. These patterns were obtained from a full-scale via layer containing over 1.9×10^6 vias, where each pixel represents an area of 1 nm^2. Our proposed workflow is not exclusive to via layer pattern. Metal layer patterns also holds repetition property, but have continuity requirement: The mask of a metal component on the stitching area shall be continuous and smooth. Adaptive OPC flow can be extended to metal layer OPC if such corner cases are handled.

For the training dataset of our ML-Solver, we randomly sliced 4000 patterns from the design layer. These patterns were accompanied by masks optimized using the ILT Solver. We used the same set of patterns to train the pattern classifier. Regarding the critical/noncritical labels, we directly applied lithography to these patterns and assigned labels based on the EPE values. This labeling approach is intuitive as it reflects the level of mask optimization difficulty. Regarding the training data for Metric Space embedding, we followed the steps outlined in Section IV-B. We selected 400 random anchor points and generated shifts around each anchor point within a range of ±10% of the pattern width. We ensured that the number of positive samples for each anchor point was equal to or greater than the number of anchors, guaranteeing a positive-to-negative ratio in each training batch. At the slicing stage, we set padding on the mask: boundary area (10% width) of pattern will be regarded as padding area such that it will go through lithography engine but only inner area the mask clip will be updated. In this way, we can avoid proximity effect problem on the edge. At the stitching stage, we need to overlap the mask clip on the padding area.

D. Performance Analysis

To validate the effectiveness of mask reuse, we initially observed the descending trend of EPE. In this regard, we conducted a demonstration experiment on a specific pattern, where we recorded the EPE descending trend during the iterations of ILT with a calibrated optimized mask serving as the initial state. We also recorded the trend when starting the ILT process from scratch without an initial mask for comparison purposes. As depicted in Fig. 9(a), when an initial mask is utilized, the EPE number starts at an almost optimal value of 23, and the descending trend converges after the first iteration. In contrast, when ILT is initiated from scratch without an initial mask, the EPE number begins at 37 and requires six iterations to reach the initial EPE number achieved through mask reuse. Overall, it takes 12 iterations for the ILT process to converge to an EPE of 22.

To assess the efficiency of our framework, we conducted runtime analysis experiments. As our framework employs highly efficient ML-based methods for handling noncritical patterns, our focus primarily lies on critical patterns. Fig. 9(b) provides a visual breakdown of the time taken by each step in the critical pattern OPC process within AdaOPC. It is evident that 91.7% of the runtime is dedicated to lithography and ILT OPC iterations. In contrast, the combined time overhead of pattern matching and shift calibration accounts for only 8.3% of the overall process time. This demonstrates that the impact of these steps on the entire process is negligible. Furthermore, this highlights the extensibility of our framework, allowing for the integration of new and powerful OPC tools or litho-models to further enhance speed. In addition, we considered the scenario where the pattern library grows larger. While generating a large number of "ground-truth" masks is limited by time and computational resources, we tested the speed of pattern matching with a substantial number of synthesized pattern vectors. As indicated in Table I, even with the pattern library expanded to store 10 000 patterns with a dimension of 512, the query and matching process can still be completed within 0.4 s. The time overhead is negligible in the overall process. To evaluate the convergence speed of mask updates with and without pattern matching, we conducted ten cases after inserting 800 patterns into the pattern library. The results in Fig. 10 demonstrate that the average number of iterations required for mask convergence can be significantly reduced by 93.6% when pattern matching is employed.

The performance gain from the post-calibration stage is listed in the last three columns on the right in Table II. We set our SOTA baseline as AdaOPC [33] with matching algorithm updated to the latest 2024 version of Faiss [36].

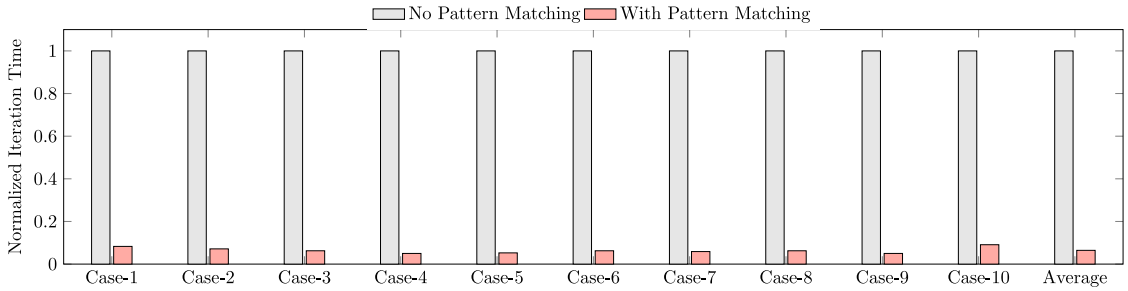

Fig. 10. Convergence speed of mask optimization with and without pattern matching. In order to clearly demonstrate the acceleration ratio, we normalized the time of mask optimization without pattern matching to 1. During optimization without pattern matching, the initial state of the mask was not calibrated. In the case of optimization with pattern matching, we utilized the calibrated mask as the initial state.

TABLE II
COMPARISONS OF BASELINE APPROACHES

Case	DAMO-DGS [12]			ILT-GPU [19]			AdaOPC [33]			AdaOPC [33]+Faiss [36]			Ours		
ID	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)	#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	22	23232	5.35	21	23492	2.42
2	26	26729	5.26	25	26762	48.5	24	26580	5.41	24	26580	5.26	22	26670	2.52
3	27	26938	5.22	24	26720	55.92	24	26718	5.37	24	26718	5.22	24	26720	2.61
4	36	27975	5.18	29	28127	70.57	25	27934	5.40	25	27934	5.25	25	27982	2.51
5	35	28805	5.32	30	28925	66.89	30	28927	5.44	30	28927	5.29	27	29128	2.67
6	30	26960	5.31	25	26762	55.81	24	26775	5.38	24	26775	5.23	24	26615	2.75
7	33	26382	5.23	28	26453	59.47	28	26281	5.43	28	26281	5.28	26	26327	2.56
8	32	30646	5.38	25	29450	54.88	27	29341	5.42	27	29341	5.27	27	29108	2.66
9	25	24054	5.25	24	24053	70.62	23	24022	5.43	23	24022	5.28	22	24111	2.63
10	24	21939	5.29	23	21701	37.59	22	21644	5.53	22	21644	5.38	21	21679	2.75
Avg.	29.0	26375	5.26	25.6	26228	56.14	24.9	26145	5.43	24.9	**26145**	5.28	**23.9**	26183	**2.61**
Ratio	1.165	1.009	0.997	1.028	1.003	10.637	1.000	1.000	1.028	1.000	**1.000**	1.000	**0.959**	1.001	**0.495**

TABLE III
ABLATIVE STUDY ON ERM AUGMENTATION

Selector Training	Error Rate
w.o augmen.	1.9%
w. augmen.	0.2%

We can tell that the average EPE error is 23.9 with the help of halo weight filtering on the loss during the final restoration process, which is even further 4% reduced from the result of conference from SOTA baseline. Table III is the ablative study on the solver selection accuracy with or without the data augmentation with ERM objective. The accuracy is improved from 98.1% to 99.8%. The error rate is reduced by 89%. With our newly implemented GPU-efficient lithography system and ILT process, the overall runtime is reduced by 52% compared to the conference version.

Finally, to evaluate the overall performance of our AdaOPC framework, we compared it with two baseline methods used for critical and noncritical patterns in different branches. After inserting 800 patterns with optimized masks into the pattern library, we randomly tested ten patterns. The results in Table II demonstrate that AdaOPC achieves comparable performance in terms of EPE and PV Band when compared to the ILT-GPU approach, but with a remarkable 20× acceleration and no loss in accuracy. Furthermore, compared to DAMO-DMG with 1 round of lithography verification, AdaOPC exhibits significantly superior performance while maintaining comparable speed. It is important to note that, in our case, the embedding vector dimension is 256. As indicated in Table I, even with the pattern library expanded to include 10 000 patterns, the matching time remains around 0.2 s, which is marginal compared to the overall OPC process time shown in Table II. In summary, the AdaOPC framework achieves a dual optimization of performance and speed, outperforming the baseline methods in accuracy and efficiency.

VIII. CONCLUSION

In this research paper, we introduced a self-adaptive OPC framework tailored for mask optimization in real designs. The framework leverages a comprehensive analysis of the design's characteristics. We proposed an extensible OPC solver selector that intelligently chooses an appropriate solver based on pattern complexity. Furthermore, we developed a dynamic pattern library that enables the reuse of optimized masks for repeating patterns with identical geometric shapes. To facilitate efficient pattern matching, we employed supervised contrastive learning to embed patterns into vectors. Additionally, we devised a graph-based search strategy to accelerate the pattern matching process. Furthermore, we validated the reusability of masks by demonstrating the pattern shift equivariance property. We also introduced a practical shift calibration tool to address any shifts that may occur. In addition to the conference version, we also add a halo-weighting strategy to conquer the sizing problem in via-layer. Through extensive experiments, we demonstrated that our framework achieves a co-optimization of OPC speed and robustness for real design patterns.

REFERENCES

[1] D. Z. Pan, B. Yu, and J.-R. Gao, "Design for manufacturing with emerging nanolithography," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 32, no. 10, pp. 1453–1472, Oct. 2013.

[2] J.-S. Park et al., "An efficient rule-based OPC approach using a DRC tool for 0.18/spl mu/m ASIC," in *Proc. IEEE Int. Symp. Quality Electron. Design (ISQED)*, 2000, pp. 81–85.

[3] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *Proc. IEEE/ACM Design, Autom. Test Eur. (DATE)*, 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 Trans. Comput. Aided Design Integr. Circuits Syst. (TCAD)*, vol. 35, no. 8, pp. 1345–1357, Aug. 2016.

[5] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical Bayes model," in *Proc. 28th Opt. Microlithogr.*, 2015, pp. 1–10.

[6] A. Poonawala and P. Milanfar, "Mask design for optical microlithography an inverse imaging problem," *IEEE Trans. Image Process.*, vol. 16, pp. 774–788, 2007.

[7] Y. Ma et al., "A unified framework for simultaneous layout decomposition and mask optimization," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5069–5082, Dec. 2020.

[8] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. IEEE/ACM Design, Autom. Test Eur. (DATE)*, 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 Trans. Comput., Aided Design Integr. Circuits Syst. (TCAD)*, vol. 39, no. 10, pp. 2822–2834, Oct. 2020.

[10] B. Jiang, H. Zhang, J. Yang, and E. F. Young, "A fast machine learning-based mask printability predictor for OPC acceleration," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 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 Trans. Comput., Aided Design Integr. Circuits Syst. (TCAD)*, vol. 39, no. 10, pp. 2849–2859, Oct. 2020.

[12] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE Trans. Comput., Aided Des. Integr. Circuits Syst. (TCAD)*, vol. 41, no. 9, pp. 3118–3131, Sep. 2022.

[13] Z. Wang et al., "DiffPattern: Layout pattern generation via discrete diffusion," in *Proc. ACM/IEEE Des. Autom. Conf. (DAC)*, 2023, pp. 1–6.

[14] Z. Pei, W. Zhao, Z. He, and B. Yu, "Bit-Level quantization for efficient layout hotspot detection," in *Proc. Int. Symp. Electron. Des. Autom. (ISEDA)*, 2023, pp. 465–470.

[15] W. Zhao et al., "A high-performance accelerator for super-resolution processing on embedded GPU," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3210–3223, Oct. 2023.

[16] 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 *Proc. IEEE/ACM Int. Conf. Comput., Aided Design (ICCAD)*, 2020, pp. 1–9.

[17] H. H. Hopkins, "The concept of partial coherence in optics," *Proc. Roy. Soc. London. Series A. Math. Phys. Sci.*, vol. 208, no. 1093, pp. 263–277, 1951.

[18] N. B. Cobb, *Fast Optical and Process Proximity Correction Algorithms for Integrated Circuit Manufacturing*. Berkeley, CA, USA: Univ. California, 1998.

[19] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. ACM/IEEE Des. Autom. Conf. (DAC)*, 2014, pp. 1–6.

[20] Y. Watanabe, T. Kimura, T. Matsunawa, and S. Nojima, "Accurate lithography simulation model based on convolutional neural networks," in *Proc. 30th Opt. Microlithogr.*, 2017, pp. 137–145.

[21] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "LithoGAN: End-to-end lithography modeling with generative adversarial networks," in *Proc. ACM/IEEE Des. Autom. Conf. (DAC)*, 2019, pp. 1–6.

[22] H.-C. Shao et al., "From IC layout to die photograph: A CNN-based data-driven approach," *IEEE Trans. Comput., Aided Des. Integr. Circuits Syst.*, vol. 40, no. 5, pp. 957–970, May 2021.

[23] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2223–2232.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

[25] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.

[26] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2006, pp. 1735–1742.

[27] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 3733–3742.

[28] Z. Pei, X. Yao, W. Zhao, and B. Yu, "Quantization via distillation and contrastive learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 23, 2023, doi: 10.1109/TNNLS.2023.3300309.

[29] R. D. Hjelm et al., "Learning deep representations by mutual information estimation and maximization," 2018, *arXiv:1808.06670*.

[30] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 1597–1607.

[31] P. Khosla et al., "Supervised contrastive learning," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 18661–18673.

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

[33] W. Zhao et al., "AdaOPC: A self-adaptive mask optimization framework for real design patterns," in *Proc. IEEE/ACM Int. Conf. Comput., Aided Design (ICCAD)*, 2022, pp. 1–9.

[34] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput., Aided Design (ICCAD)*, 2013, pp. 271–274.

[35] T. Ajayi and D. Blaauw, "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. Gov. Microcircuit Appl. Critical Technol. Conf.*, 2019, pp. 1–6.

[36] M. Douze et al., "The Faiss library," 2024, *arXiv:2401.08281*.

Wenqian Zhao received the B.Sc. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering.

His research interests include machine learning for VLSI design automation and hardware-aware deep-learning acceleration.

Xufeng Yao received the B.Eng. degree in information system and information management from Fudan University, Shanghai, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include computer vision and machine learning in EDA.

Shuo Yin received the B.Eng. degree in computer science and engineering from Beihang University, Beijing, China, in 2022. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests include programming language, compiler design for hardware, and large-scale GPU parallelization for EDA.

Yang Bai received the B.S. degree in telecommunications engineering from Xidian University, Xi'an, China, in 2017, and the master's degree in computer science from the Chinese Academy of Sciences, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests focus on the optimization for deep neural network training and inference via compilation techniques.

Ziyang Yu received the B.S. degree from the Department of Physics, University of Science and Technology of China, Hefei, China, in 2018, and the M.Phil. degree from the Department of Physics, University of Hong Kong, Hong Kong, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His current research interests include design space exploration in electronic design automation and machine learning on chips.

Yuzhe Ma (Member, IEEE) received the B.E. degree from the Department of Microelectronics, Sun Yat-sen University, Guangzhou, China, in 2016, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2020.

He is currently an Assistant Professor of Microelectronics Thrust with The Hong Kong University of Science and Technology (Guangzhou), Guangzhou. His research interests include agile VLSI design methodologies, machine-learning-aided VLSI design, and hardware-friendly machine learning.

Dr. Ma received the Best Paper Awards from ICCAD 2021, ASPDAC 2021, and ICTAI 2019, and the Best Paper Award Nomination from ASPDAC 2019.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received nine Best Paper Awards from DATE 2022, ICCAD 2021 and 2013, ASPDAC 2021 and 2012, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, and SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards. He has served as the TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TCCPS Newsletter.

Martin D. F. Wong (Fellow, IEEE) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign (UIUC), Champaign, IL, USA, in 1981 and 1987, respectively.

He was a Faculty Member with The University of Texas at Austin (UT-Austin), Austin, TX, USA, from 1987 to 2002 and UIUC from 2002 to 2018. He was a Bruton Centennial Professor of Computer Science with UT-Austin and an Edward C. Jordan Professor of Electrical and Computer Engineering with UIUC. From August 2012 to December 2018, he was the Executive Associate Dean of the College of Engineering, UIUC. In January 2019, he joined The Chinese University of Hong Kong, Hong Kong, as the Dean of Engineering and a Choh-Ming Li Professor of Computer Science and Engineering. He has published around 500 papers and graduated over 50 Ph.D. students in electronic design automation (EDA). His main research interest is in EDA.

Prof. Wong is a Fellow of ACM.