DAMO: Deep Agile Mask Optimization for Full-Chip Scale

Guojin Chen, Wanli Chen, Qi Sun, *Graduate Student Member, IEEE*, Yuzhe Ma, *Member, IEEE*, Haoyu Yang, and Bei Yu, *Member, IEEE*

***Abstract*—Continuous scaling of the very-large-scale integration system leaves a significant challenge on manufacturing; thus optical proximity correction (OPC) is widely applied in conventional design flow for manufacturability optimization. Traditional techniques conduct OPC by leveraging a lithography model but may suffer from prohibitive computational overhead. In addition, most of them focus on optimizing a single and local clip instead of addressing how to tackle the full-chip scale. In this article, we present DAMO, a high-performance and scalable deep-learning-enabled OPC system for full-chip scale. It is an end-to-end mask optimization paradigm that contains a deep lithography simulator (DLS) for lithography modeling and a deep mask generator (DMG) for mask pattern generation. Moreover, a novel layout splitting algorithm customized for DAMO is proposed, composed of DBSCAN clustering and KMeans++ clustering, to handle the full-chip OPC problem. Further, graph-based computation and parallelism techniques are proposed to deploy our GPU algorithms to accelerate computations. Extensive experiments show that DAMO outperforms state-of-the-art OPC solutions in both academia and industrial commercial toolkit.**

***Index Terms*—Deep learning, design for manufacture, mask optimization, optical proximity correction (OPC).**

I. Introduction

CONTINUOUSLY shrinking down of the very large-scale integration (VLSI) system has brought inevitable lithograph proximity effects, resulting in degradation on manufacturing yield [1]. Optical proximity correction (OPC) compensates for lithography proximity effects by adding assistant features and moving design edge segments inward or outward [2]. Mainstream OPC solutions include rule-based OPC [3], model-based OPC [4]–[6], inverse lithography technique (ILT)-based OPC [7], [8], and machine/deep-learning-based OPC [9]–[11].

Kuang *et al.* [4] presented a model-based OPC for faster convergence and achieved good edge placement error (EPE) with minor PV Band overhead using multistage SRAF insertion and OPC. Gao *et al.* [7] tackled the mask optimization problem by solving an ILT formulation, which descends the gradient of wafer-target error over input masks. The pixel-based optimization of the ILT solution makes them robust to process variations. The generality of ILT also enables simultaneous mask optimization and layout decomposition as introduced in [8] and [12]. These methods, to some extent, improve OPC quality, robustness, and efficiency.

The remarkable development of machine learning algorithms has demonstrated the potential of applying artificial intelligence to benefit modern OPC flows. On the one hand, machine-learning-guided mask optimization targets to generate masks that are close to the optimal status directly. Only fewer fine-tune steps using traditional OPC engines are required to obtain the final mask. Yang *et al.* [9] proposed GAN-OPC, which grasps the advantage of generative machine learning models that can learn a design-to-mask mapping and provides better initialization of the ILT engine. On the other hand, machine-learning-based lithography simulation aims to speed up OPC flows by replacing costly lithography simulation with efficient learning models. Jiang *et al.* [10] applied an XGBoost [13] learning model to predict EPE at certain OPC control points that can guide the adjustment of shape edges. Ye *et al.* [14] proposed LithoGAN to build a generative learning model that directly predicts lithography contours instead of predicting wafer image errors. However, LithoGAN only targets a single shape within a clip, which strictly limits its usage in general OPC tasks.

There are several issues in previous methods. First, the model-based/ILT inevitably methods require massive calls of the costly lithography simulation and the mask optimization, both of which are time consuming. Second, all previous machine-learning-guided OPC works limit the single-clip input layout to low-resolution images (e.g., 256×256-pixel images). They all exhibit drawbacks that still have to go through traditional OPC engines in the final steps due to the low-resolution limits. Since the resolution loss is intolerable in OPC, the usage scenarios of previous work in machine-learning-guided OPC are limited. And worse still, the machine-learning-based single-clip OPC is not practical. Third, although various methods have been proposed, most of them focused on optimizing a given single clip, but rarely discussed how to tackle the OPC problem in a view of full-chip scale. For full-chip OPC tasks, the most significant barrier to conventional methods is the runtime overhead. Pang *et al.* [15]

Manuscript received 9 March 2021; revised 12 July 2021; accepted 8 September 2021. Date of publication 29 September 2021; date of current version 22 August 2022. This work was supported in part by the Research Grants Council of Hong Kong SAR under Project CUHK14208021 and Project CUHK14209420. The preliminary version has been presented at the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2020. This article was recommended by Associate Editor L. Behjat. *(Corresponding author: Bei Yu.)*

Guojin Chen, Wanli Chen, Qi Sun, Yuzhe Ma, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Haoyu Yang is with the ASIC and VLSI Research Group, NVIDIA Research, NVIDIA, Austin, TX 78717 USA.

Digital Object Identifier 10.1109/TCAD.2021.3116511

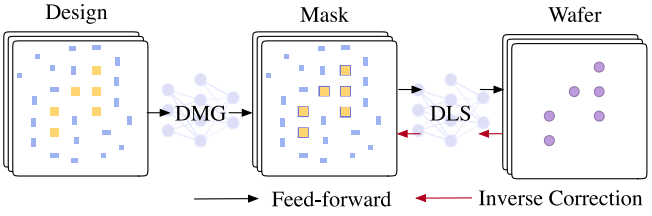

Fig. 1. Overview of our proposed DAMO framework, which consists of two deep networks: 1) DMG and 2) DLS. The OPC process is completed by utilizing the inverse correction gradient backpropagated from the DLS stage (red arrows).

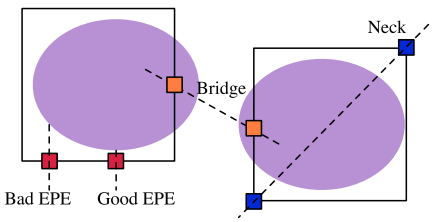

Fig. 2. Different types of defects. Same lithography images result in different EPE violations due to different choices of measurement points. Some defects are not detectable by merely checking EPE.

presented D2S to create full-chip ILT in a single day with giant GPU/CPU pairs, which consumes many resources on the hand-crafted hardware and software. To the best of our knowledge, the learning-based methods have not achieved any progress on full-chip mask optimization due to the dataset limitation and the low wafer pattern fidelity.

To address these concerns, we present DAMO, a unified OPC engine that is equipped with high-resolution GANs for full-chip scale. Deep convolution GANs (DCGAN) [16] have been demonstrated to be successful in generating high-resolution images. In DAMO, we designed DCGAN-HD, customized from DCGAN with a high-resolution generator and multiscale discriminators with perceptual loss. Then, we design a deep lithography simulator (DLS) based on DCGAN-HD that takes the input of the mask and generates the lithography contours faster with similar contour quality compared to the legacy lithography simulation process. The design of DLS also enables a unified neural network-based OPC framework where another deep mask generator (DMG) engine is trained along with the gradient backpropagated from DLS, as shown in Fig. 1. We further propose a stitchless full-chip splitting algorithm, with which we can perform full-chip OPC tasks efficiently with a few GPU resources and much faster speeds. Our main contributions are as follows.

1) We design DCGAN-HD, a very competitive high-resolution feature extractor (1024×1024) by redesign the generator and discriminator of DCGAN.
2) We build up DLS and DMG based on DCGAN-HD. DLS is expected to conduct high-resolution lithography simulations. By training along with the inverse correction from DLS, DMG can directly generate high-quality masks.
3) We develop an efficient stitchless full-chip splitting algorithm, composed of DBSCAN and KMeans++ clustering algorithms, to apply DAMO on a layout of any size.
4) We propose to use a graph-based computation technique and parallelism technique to accelerate the computations of DBSCAN and KMeans++ on GPU, respectively. Besides, the proposed DCGAN-HD is also deployed by using TensorRT, which results in much faster inferences.
5) We compare our proposed framework with state-of-the-art commercial tool Calibre [17]: $5\times$ speed-up in single-clip OPC tasks and $1.3\times$ acceleration in full-chip OPC tasks while maintaining even better solution quality.

The remainder of this article is organized as follows. Section II introduces terminologies and evaluation metrics related to this work. Section III details the proposed DAMO architecture. Section IV shows the data preparation and DAMO training procedure, while Section V provides the full-chip splitting algorithm and the acceleration techniques. Section VI details experimental results and followed by a conclusion in Section VII.

II. Preliminaries

In this section, we will introduce some concepts and background related to this work and the problem formulation.

A. cGAN Basis

cGAN is short for conditional generative adversarial networks [18], [19], which resembles classical GANs [20] that consist of a generator and a discriminator. The generator is trained to generate patterns that follow some distribution such that the discriminator cannot identify whether these data come from the generator or the training dataset. cGAN differs from GANs by certain constraints such that inputs and outputs of the generator can have stronger beneath connections. Representative cGAN applications in VLSI include GAN-OPC [9] and LithoGAN [14]. The former is designed for layout mask synthesis and the latter focuses on lithography contour prediction of the single via/contact shapes.

B. Problem Formulation

We introduce the following terms and evaluation metrics for the DAMO framework.

Definition 1 (mIoU): Given two shapes P and G, the IoU between P and G is $IoU(P, G) = P \cap G / P \cup G$. The mIoU is mean IoU.

Definition 2 (Pixel Accuracy): Pixel accuracy (pixAcc) is defined as the percentage of pixels correctly classified on an image.

The mask quality is evaluated through the fidelity of its wafer image with respect to its target image. As illustrated in Fig. 2, EPE, bridge, and neck are three main types of defect detectors that are adopted in a layout printability estimation flow. EPE is calculated by the distance from the measurement sites on target edges to lithography contours. The neck defect is the error of critical dimensions of lithography contours compared to target patterns, while the bridge detector aims to find unexpected short of wires. Both neck and bridge defects can

appear in any direction. Because EPE violations could happen with good critical dimension and neck or bridge occurs with small EPE, none of these defect types individually can be an ideal representation of mask printability. Considering the objective of mask optimization is to make sure the remaining patterns after the lithography process is as close as target patterns, we have two evaluation metrics to measure mask quality following [9]. The squared L_2 error measures the quality of a mask under nominal process conditions, while PV Band measures the robustness of the generated mask under variations.

Definition 3 (Squared L_2 Error): Let $\boldsymbol{w}$ and $\boldsymbol{y}$ as design image and wafer image, respectively, the squared L_2 error is calculated by $||\boldsymbol{w}-\boldsymbol{y}||_2^2$.

Definition 4 (PV Band): Given the lithography simulation contours under a set of process conditions, the PV Bands are the area among all the contours under these conditions.

With these definitions and evaluation metrics, the problem of mask optimization is defined as follows.

Problem 1 (Mask Optimization): Given a design image $\boldsymbol{w}$, the objective of mask optimization is generating the corresponding mask $\boldsymbol{x}$ such that remaining patterns $\boldsymbol{y}$ after the lithography process is as close as $\boldsymbol{w}$ or, in other words, minimizing PV Band and squared L_2 error of lithography images.

III. DAMO Framework

The architecture overview of DAMO is illustrated in Fig. 3. As the first part of DAMO, DLS aims to conduct an efficient and high-quality lithography simulation with the generative neural network model. However, general cGAN is not directly eligible for such a purpose due to potential contour-design misalignment and low accuracy. Although LithoGAN [14] tries to alleviate the problem by embedding coordinate inputs, the scenario of application is strictly limited to a single via/contact shape, which is not practical in most cases. Therefore, DLS is developed as a customized cGAN for general-purpose lithography contour prediction tasks.

DMG is the second part of DAMO, which shares the identical architecture with DLS. The forward lithography process can be described with the following equation:

$$\boldsymbol{Z} = f(\boldsymbol{M}). \tag{1}$$

The traditional ILT tries to obtain the optimal mask $\boldsymbol{M}_{\text{opt}}$ based on the given lithography model, which is presented as

$$\boldsymbol{M}_{\text{opt}} = f^{-1}(\boldsymbol{Z}_t) \tag{2}$$

where $\boldsymbol{Z}_t$ is the design pattern and $\boldsymbol{M}_{\text{opt}}$ is the optimized mask with OPC. In DAMO, we regard DLS as f in (1). However, different masks may yield the same result, thus (2) is an ill-posed problem. The previous mask optimizer GAN-OPC [21] generates masks by using cGAN to learn the mapping between the design and the mask pattern. Inspired by conventional ILT, our DMG steps further by not only learning mask patterns from training datasets but also being optimized by gradient backpropagated from the pretrained DLS. After training, the generator of DMG performs inference to generate the solutions.

A. Improving Accuracy by Higher Resolution

Different from synthesizing photo-realistic images in computer vision tasks, the OPC task using generative models has its own properties. Intuitively, the layout in the OPC task has simpler patterns (mostly rectangles) but higher precision demands compared with image translation tasks. Moreover, the inputs of traditional image generation tasks are fixed-size images whose width or height is barely more than 2048 pixels. However, layouts contain thousands of via/contacts or SRAF patterns, whose area can reach more than 100×100 um^2. Previous work GAN-OPC [9] converts 1000×1000 nm^2 layout to 256×256 pixel images, which means 1-pixel shift error will cause an 8-nm shift in the output layout, making the results vulnerable for the industrial OPC tasks. To eliminate image transformation error, we set the input resolution of our model to be 1024×1024 pixels to contain the full 1024×1024 nm^2 layout. Combined with the window splitting algorithm, which will be introduced in Section VI, the DAMO framework can process input layouts of any size, even the large full-chip layouts.

It is known that the adversarial training might be unstable and hard to converge for high-resolution image generation tasks, as mentioned in [16], [22], and [23]. Therefore, we present DCGAN-HD, a new conditional GANs model qualified with high-resolution input images, which is the basic architecture of DLS and DMG.

B. DCGAN-HD: Solution for High Resolution

Previous work GAN-OPC is a conditional GAN framework for design to mask translation which consists of a generator G and a discriminator D. It adopts U-Net [24] as the generator with the input resolution of 256×256, We tested the GAN-OPC framework directly on high-resolution images and found the training is unstable and the generated results usually became empty. DCGAN [16] is one of the popular and successful network designs for cGAN allowing for higher resolution and deeper models. Based on DCGAN we present DCGAN-HD, a robust high-resolution conditional GAN model consisting of a newly designed generator, multiscale discriminators, and a novel adversarial loss function. The architecture is illustrated in Fig. 3. The detailed architecture of DCGAN-HD is listed in Table I.

1) High-Resolution Generator for DCGAN-HD: The left part of Fig. 3 shows the high-resolution generator. In the DLS part, the generator of DCGAN-HD resembles lithography simulation, which requires mask-to-wafer mapping. With the gradient backpropagated from DLS, in the DMG part, the generator focuses on synthesizing the mask patterns from design and SRAF pattern groups.

UNet++ Backbone: Previous work [9] and [14] adopt traditional UNet [24] for mask generation. Input features are downsampled multiple times. With the decrease in feature resolution, it is easier for a network to gather high-level features such as context features, while low-level information such as the position of each shape becomes harder to collect. However, in OPC tasks, low-level information matters more than in common computer vision tasks. For example,

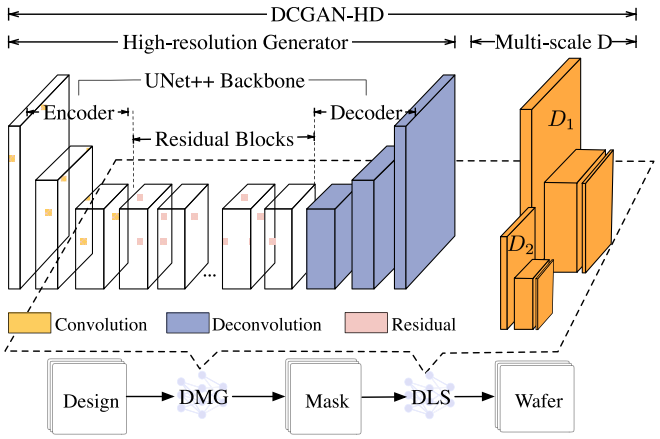

Fig. 3. Architecture of DCGAN-HD with high-resolution generator and multiscale discriminators, used in both DMG and DLS.

TABLE I
ARCHITECTURE OF DCGAN-HD

High Resolution Generator for DCGAN-HD			
Layer	Filter	Stride	Output Size
Input	—	—	1024×1024×3
Conv-BN-ReLU	7×7	1	$1024 \times 1024 \times 32$
Conv-BN-ReLU	3×3	2	$512 \times 512 \times 64$
Conv-BN-ReLU	3×3	2	$256 \times 256 \times 128$
Conv-BN-ReLU	3×3	2	$128 \times 128 \times 256.$
Conv-BN-ReLU	3×3	2	$64 \times 64 \times 512$
Conv-BN-ReLU	3×3	2	$32 \times 32 \times 1024$
Residual Block 1...9	3×3	1	$32 \times 32 \times 1024$
Deconv-BN-ReLU	3×3	2	$64 \times 64 \times 512$
Deconv-BN-ReLU	3×3	2	$128 \times 128 \times 256$
Deconv-BN-ReLU	3×3	2	$256 \times 256 \times 128$
Deconv-BN-ReLU	3×3	2	$512 \times 512 \times 64$
Deconv-BN-ReLU	7×7	1	$1024 \times 1024 \times 3$
Discriminator D_1			
Input	—	—	$1024 \times 1024 \times 6$
Conv-BN-LeakyReLU	4×4	2	$512 \times 512 \times 64$
Dropout (rate=0.5)	—	1	$512 \times 512 \times 64$
Conv-BN-LeakyReLU	4×4	1	$512 \times 512 \times 128$
Dropout (rate=0.5)	—	1	$512 \times 512 \times 128$
Conv-BN-LeakyReLU	4×4	1	$512 \times 512 \times 1$
Discriminator D_2			
Input	—	—	$512 \times 512 \times 6$
Conv-BN-LeakyReLU	4×4	2	$256 \times 256 \times 64$
Dropout (rate=0.5)	—	1	$256 \times 256 \times 64$
Conv-BN-LeakyReLU	4×4	1	$256 \times 256 \times 128$
Dropout (rate=0.5)	—	1	$256 \times 256 \times 128$
Conv-BN-LeakyReLU	4×4	1	$256 \times 256 \times 1$

the shape and relative distance of design or SRAF patterns must remain unchanged after the deep mask optimization or deep lithography process. The number and relative distance of via patterns in an input layout have a crucial influence on the result. The features of OPC datasets determine the vital importance of the low-level features. UNet++ [25] is hence proposed for better feature extraction by assembling multiple UNet that have different numbers of downsampling operations. It redesigns the skip pathways to bridge the semantic gap between the encoder and decoder feature maps, contributing to the more accurate low-level feature extraction. The dense skip connections on UNet++ skip pathways improve gradient flow in high-resolution tasks. It is an ideal solution to low-level and high-level feature capturing. Although UNet++ has a better performance than UNet, it is not qualified to be the generator of DCGAN-HD. First, we replace all pooling layers with stride convolution. Meanwhile, all bilinear upsample layers are switched into transpose convolution. This amendment makes the entire network different. Second, batch normalization is applied after each convolution layer, which helps the convergence of the neural network. Finally, the Tanh activation function is used after the output layer. For further improvement, we manipulate the UNet++ backbone with the guidelines suggested in DCGAN [16]. We will show later that our high-resolution generator outperforms UNet and UNet++ by a large margin.

Residual Blocks: Most importantly, following the settings from Johnson *et al.* [26], a set of residual blocks are added at the bottleneck of UNet++, which has been proven successful in style transfer and high-resolution image synthesis tasks. Since most structures are shared in output and input images (design and SRAFs), residual connections make it easy for the network to learn the identity function, which is appealing in the mask generation process. Specifically, we use nine residual blocks, each containing two 3×3 convolution layers and batch normalization layers.

2) Multiscale Discriminators for DCGAN-HD: The high-resolution input also imposes a critical challenge to the discriminator design. A simple discriminator that only has three convolutional layers with LeakyReLU [27] and Dropout [28] is presented. Since the patterns in OPC datasets have simple and homogeneous distribution, a deeper discriminator has a higher risk of overfitting. Therefore, we simplify the discriminator by reducing the depth of the neural network. Meanwhile, a dropout layer is attached after each convolutional layer. We use 3×3 convolution kernels in the generator for parameter-saving purposes and 4×4 kernels in the discriminator to increase receptive fields.

However, during training, we find that the simple discriminator fails to distinguish between the authentic and the synthesized images when more via patterns occur in a window. Because when the number of via reaches 5 or 6 in a window, the via patterns will have a more significant impact on each other and the features become more complicated. Inspired by Wang *et al.* [23] in pix2pixHD, we design multiscale discriminators. Different from pix2pixHD [23] that using three discriminators, our design uses two discriminators that have an identical network structure but operate at different image scales, which is named $D1$ and $D2$, as shown in the right part of Fig. 3. Specifically, the discriminators $D1$ and $D2$ are trained to differentiate real and synthesized images at the two different scales, 1024×1024 and 512×512, respectively, which helps the training of the high-resolution model easier.

In our tasks, the multiscale design also shows its strengths in flexibility. For example, when the training set has only one via in a window, we can use only $D1$ to avoid overfitting and reduce the training time.

3) Perceptual Loss: Instead of using per-pixel loss such as L_1 loss or L_2 loss, we adopt the perceptual loss which has been proven successful in style transfer [26], image super-resolution and high-resolution image synthesis [23]. A per-pixel loss function is used as a metric for understanding differences between input and output on a pixel level. While the function is valuable for understanding interpolation on a pixel level, the process has drawbacks. For example, as stated in [26], consider two identical images offset from each other by one pixel; despite their perceptual similarity, they would be very different as measured by per-pixel losses. More than that, previous work [16] shows L_2 Loss will cause blur on the output image. Different from per-pixel loss, the perceptual loss function in (3) compares ground-truth image $\boldsymbol{x}$ with generated image $\hat{\boldsymbol{x}}$ based on the generated results from pretrained convolutional neural networks Φ, which is ideal in the DAMO framework. In the DLS part, since the wafer pattern is not a regular circle, it is meaningless to fit the exact border of a wafer on the pixel level. The ultimate goal is to generate a better mask with a higher perceptual quality wafer, reflected in less L_2 error and smaller PV Band

$$\mathcal{L}_{L_P}^{G,\Phi}(\boldsymbol{x},\hat{\boldsymbol{x}}) = \mathcal{L}_{L_1}\big(\Phi(\boldsymbol{x}),\Phi(\hat{\boldsymbol{x}})\big) = \mathbb{E}_{\boldsymbol{x},\hat{\boldsymbol{x}}}\big[\|\Phi(\boldsymbol{x}) - \Phi(\hat{\boldsymbol{x}})\|_1\big]. \tag{3}$$

4) SSIM Loss: Recently, the structural similarity index method (SSIM) is actively explored as a feasible alternative for the pixel-level loss. SSIM was first introduced in [29], which is significantly capable of identifying the structural information from the scene. In the enhanced DAMO (EDAMO) framework, besides the perceptual loss, we also apply the SSIM loss to capture the structural and geometric information to generate high-fidelity wafer patterns. The SSIM metric extracts three critical features from a given image: 1) luminance; 2) contrast; and 3) structure. Then, the SSIM loss is given by

$$\mathcal{L}_{\text{SSIM}}(\boldsymbol{x},\hat{\boldsymbol{x}}) = 1 - \frac{(2\mu_x\mu_{\hat{x}} + C_1)(2\sigma_{x\hat{x}} + C_2)}{(\mu_x^2 + \mu_{\hat{x}}^2 + C_1)(\sigma_x^2 + \sigma_{\hat{x}}^2 + C_2)} \tag{4}$$

where the luminance $\mu_x = (1/N)\sum_{i=1}^{N} x_i$ is measured by the averaging over all the pixel values in image x. The contrast metric is calculated by the standard deviation of all the pixel values, $\sigma_x = \sqrt{[1/(N-1)]\sum_{i=1}^{N}(x_i - \mu_x)^2}$. In discrete form, $\sigma_{x\hat{x}}$ can be estimated as $\sigma_{x\hat{x}} = [1/(N-1)]\sum_{i=1}^{N}(x_i - \mu_x)(\hat{x}_i - \mu_{\hat{x}})$. $C_1 = k_1 L$ and $C_2 = k_2 L$, where L is the dynamic range for pixel values. In our implementation, $k_1 = 0.01$ and $k_2 = 0.03$. Instead of applying the SSIM loss globally, we use local mean SSIM (MSSIM) following the instructions in [29]:

$$\mathcal{L}_{\text{MSSIM}}(\boldsymbol{x},\hat{\boldsymbol{x}}) = \frac{1}{M}\sum_{j=1}^{M}\mathcal{L}_{\text{SSIM}}(\boldsymbol{x}_j,\hat{\boldsymbol{x}}_j) \tag{5}$$

where M is the number of local windows in the image.

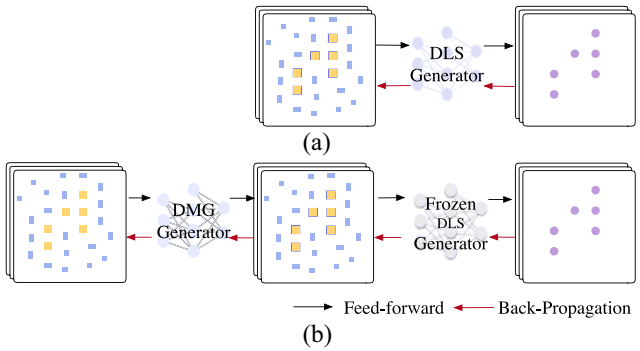

Fig. 4. Overall training of DAMO. (a) Training DLS in the first stage. (b) Training DMG with fixed DLS generator in the second stage.

IV. Data Preparation and Training

In order to collect sufficient data for training, we develop a data generation pipeline that can generate infinite training data, with which our DCGAN-HD can be fully utilized to simulate the lithography process and generate high-quality mask patterns. The overall training procedure of DAMO can be divided into two parts which are depicted in Fig. 4.

A. Building Training Set From Scratch

It takes five steps to generate a training image, including design generation, SRAF insertion (with design rule checking), OPC, lithography simulation, and layout to image transformation.

Design a Design Pattern: Via patterns are obtained under the following constraints using a layout pattern generator [30]. First, all via patterns (70×70 nm^2) are restricted in a 1024×1024 nm^2 window. Second, by changing the via density, we can control the number of via patterns in a single window. The via patterns are grouped evenly by the via numbers for reducing the bias caused by the random distribution of training set.

SRAF Insertion and DRC: Mentor Calibre [17] is applied to do the SRAF insertion and design rule checking. Note that the via patterns may appear on any position inside the design area, Calibre will create a new design layer centered at the center of the via patterns when doing SRAF insertion. This makes the model easier to learn because all via patterns locate near the center of the image. In addition, it gives us guidance to develop further speed-up algorithms. Since the design area is 1024×1024 nm^2, it is possible that a few of SRAF patterns will be outside the design area when there are more than 2 via patterns. A larger window of 2048×2048 nm^2 will be used to capture all the SRAF patterns, which shares the same center as the design window.

OPC, Litho-Simulation, and Image Generation: We use masks and wafer patterns generated by Calibre as ground truth. All the via patterns, SRAFs, and masks are discriminated by RGB channels. Two sets of paired data are required for training. Mask–wafer pairs are generated to train DLS. After that, we align design–mask–wafer data for the OPC process. The obtained clips of size 2048×2048 nm^2 are converted into images with 2048×2048 pixels where 1 nm represents one

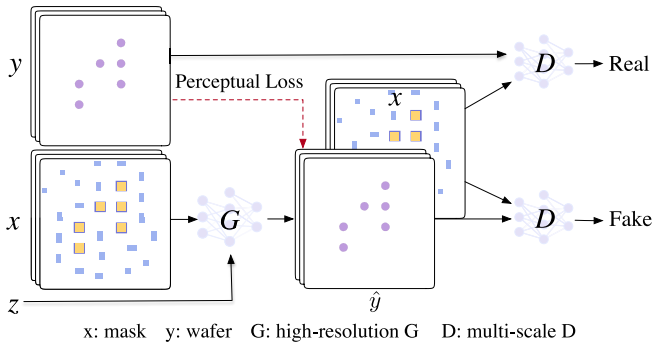

Fig. 5. Training details of DLS, where the input images are mask–wafer pairs.

pixel. All the 2048 × 2048 pixels images will be centrally cropped into 1024 × 1024 pixels images where the design window locates before training. After training, the generated 1024 pixels images will be attached at the center of the SRAF clip layer to form a 2048 × 2048 nm^2 layout before testing using Calibre. The crop-then-recover strategy saves the computational cost and improves the accuracy by focusing on the mask generation.

B. Training of DLS

Fig. 5 shows the training process of our deep lithography simulator. As a customized design of cGAN, DLS is trained in an alternative scheme using paired mask image $\boldsymbol{x}$ and wafer image $\boldsymbol{y}$ obtained from Mentor Calibre. z indicates randomly initialized images.

The objectives of DLS include training the generator G that produces fake wafer images $G(\boldsymbol{x}, z)$ by learning the feature distribution from $\boldsymbol{x}$–$\boldsymbol{y}$ pairs and training the discriminators D_1 and D_2 to identify the paired $(\boldsymbol{x}, G(\boldsymbol{x}, z))$ as fake. This motivates the design of the DLS loss function. The first part of the loss function comes from vanilla GAN that allows the generator and the discriminator interacting with each other in an adversarial way

$$\mathcal{L}_{\text{cGAN}}(G, D) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\big[\log D(\boldsymbol{x}, \boldsymbol{y})\big] + \mathbb{E}_{\boldsymbol{x},z}\big[\log(1 - D(\boldsymbol{x}, G(\boldsymbol{x}, z))\big]. \quad (6)$$

Combined with our multiscale discriminators described in Section III-B2, (6) can be modified as

$$\sum_{k=1,2} \mathcal{L}_{\text{cGAN}}\big(G_{\text{DLS}}, D_{\text{DLS}_k}\big) = \sum_{k=1,2} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}\big[\log D_{\text{DLS}_k}(\boldsymbol{x}, \boldsymbol{y})\big] + \mathbb{E}_{\boldsymbol{x},z}\big[\log\big(1 - D_{\text{DLS}_k}(\boldsymbol{x}, G_{\text{DLS}}(\boldsymbol{x}, z))\big)\big] \quad (7)$$

where D_{DLS_k} is the kth discriminator of DLS. In the DLS design, the perceptual loss is added to the objective, we denote $\hat{\boldsymbol{y}}$ as $G(\boldsymbol{x}, z)$ and loss network Φ is a pretrained VGG19 on ImageNet. The perceptual loss is given by

$$\mathcal{L}_{L_P}^{G_{\text{DLS}},\Phi}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{j=1\ldots5} \mathcal{L}_{L_1}\big(\phi_j(\boldsymbol{y}), \phi_j(\hat{\boldsymbol{y}})\big) = \sum_{j=1\ldots5} \mathbb{E}_{\boldsymbol{y},\hat{\boldsymbol{y}}}\big[\big\|\phi_j(\boldsymbol{y}) - \phi_j(\hat{\boldsymbol{y}})\big\|_1\big] \quad (8)$$

where ϕ_j is the feature representation on the jth layer of the pretrained VGG19 Φ. To capture more structural information, we also add the SSIM loss for DLS training in the EDAMO framework. By combining (5), (7), and (8)

$$\mathcal{L}_{\text{DLS}} = \sum_{k=1,2} \mathcal{L}_{\text{cGAN}}\big(G_{\text{DLS}}, D_{\text{DLS}_k}\big) + \lambda_0 \mathcal{L}_{L_P}^{G_{\text{DLS}},\Phi}(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \alpha_0 \mathcal{L}_{\text{MSSIM}}(\boldsymbol{y}, \hat{\boldsymbol{y}}). \quad (9)$$

C. Training of DAMO

Here, we introduce the overall training procedures of the whole framework. The first training step is illustrated in Fig. 4(a), which is focusing on DLS. The proposed DLS is expected to predict wafer image with higher precision compared with traditional cGAN. After the training of DLS, all parameters in its generator are frozen.

The second training step is illustrated in Fig. 4(b), which is focusing on DMG. DMG has the same architecture as DLS developed for DAMO training. In this stage, training data are switched to design–mask–wafer pairs. We use the design–mask to train DMG, obtaining an initial solution. The objective of DMG is shown in (11), where $\boldsymbol{x}$ represents the ground-truth mask, $\boldsymbol{w}$ is the corresponding design, and z_0 is the image with random values. G_{DMG} and D_{DMG} represent the generator and discriminator of DMG. $\hat{\boldsymbol{x}}$ is the generated mask of G_{DMG}. Here, DMG shares the same architecture as DLS, which yields a similar objective as (9)

$$\sum_{k=1,2} \mathcal{L}_{\text{cGAN}}\big(G_{\text{DMG}}, D_{\text{DMG}_k}\big) = \sum_{k=1,2} \mathbb{E}_{\boldsymbol{w},\boldsymbol{x}}\big[\log D_{\text{DMG}_k}(\boldsymbol{w}, \boldsymbol{x})\big] + \mathbb{E}_{\boldsymbol{w},\boldsymbol{z_0}}\big[\log\big(1 - D_{\text{DMG}_k}(\boldsymbol{w}, G_{\text{DMG}}(\boldsymbol{w}, z_0))\big)\big]. \quad (10)$$

$$\mathcal{L}_{\text{DMG}} = \sum_{k=1,2} \mathcal{L}_{\text{cGAN}}\big(G_{\text{DMG}}, D_{\text{DMG}_k}\big) + \lambda_1 \mathcal{L}_{L_P}^{G_{\text{DMG}},\Phi}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \alpha_1 \mathcal{L}_{\text{MSSIM}}(\boldsymbol{x}, \hat{\boldsymbol{x}}). \quad (11)$$

Then, the outputs of DMG will be put into DLS for lithography simulation. After comparing the lithography results of DLS with ground truth (wafer image), DLS backpropagates the gradients to DMG, which can finally output high-quality masks after repetitive training. RGB images instead of binary images are used because we can control the gradient of design, mask, and wafer separately, which is significant for avoiding noise points. Separating the design, mask, and SRAF into different channels makes DAMO more stable and flexible because we can apply different weights on different channels. After that, DLS calculates the perceptual loss between the generated wafer and the ground-truth wafer. Finally, the gradient will be backpropagated to DMG to guide mask generation. Combining (9) with (11), the objective function of DAMO can be expressed as

$$\mathcal{L}_{\text{DAMO}} = \mathcal{L}_{\text{DMG}} + \mathcal{L}_{\text{DLS}} + \lambda_2 \mathcal{L}_{L_1}(\hat{\boldsymbol{y}}, \boldsymbol{w}_r). \quad (12)$$

We denote $\boldsymbol{w}_r$ as the via patterns (without SRAF). The last term in (12) shows the superiority of our architecture, which bridges the gap between the generated wafer $(\hat{\boldsymbol{y}})$ and target

Algorithm 1 Training Algorithm of DAMO

1: Setting λ_0 λ_1 λ_2;
2: **for** number of training iterations **do**
3: Sample design patterns $\mathcal{W} \leftarrow \{w_1, \ldots, w_b\}$;
4: Sample mask patterns $\mathcal{X} \leftarrow \{x_1, \ldots, x_b\}$;
5: Sample wafer patterns $\mathcal{Y} \leftarrow \{y_1, \ldots, y_b\}$;
6: Initialize random noise $\mathcal{Z}_0 \leftarrow \{z_{01}, \ldots, z_{0b}\}$;
7: Initialize random noise $\mathcal{Z} \leftarrow \{z_1, \ldots, z_b\}$;
8: Concatenate $\mathcal{W}$, $\mathcal{Z}_0 \leftarrow \{\{w_1, z_{01}\}, \ldots, \{w_b, z_{0b}\}\}$;
9: $\Delta \boldsymbol{W}_g \leftarrow 0$, $\Delta \boldsymbol{W}_d \leftarrow 0$;
10: **for** each $\{w_i, z_{0i}\} \in \mathcal{W}, \mathcal{Z}_0$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$ and $z_i \in \mathcal{Z}$ **do**
11: $\hat{x}_i \leftarrow G_{DMG}(w_i, z_{0i})$;
12: $\mathcal{L}_g \leftarrow -\log(D_{DMG}(w_i, \hat{x}_i)) + \lambda_0 \|\phi(x_i) - \phi(\hat{x}_i)\|_1$;
13: $\mathcal{L}_d \leftarrow \log(D_{DMG}(w_i, \hat{x}_i)) - \log(D_{DMG}(w_i, x_i))$
14: Concatenate $\{\hat{x}_i, z_i\}$ for DLS input;
15: Fix $\boldsymbol{W}_{g_{DLS}}$;
16: $w_{ri} \leftarrow$ Remove SRAF in w_i
17: $\hat{y}_i \leftarrow G_{DLS}(\hat{x}_i, z_i)$;
18: $\mathcal{L}_{g_{DLS}} \leftarrow -\log(D_{DLS}(x_i, \hat{y}_i)) + \lambda_1 \|\phi(y_i) - \phi(\hat{y}_i)\|_1$;
19: $\mathcal{L}_{d_{DLS}} \leftarrow \log(D_{DLS}(x_i, \hat{y}_i)) - \log(D_{DLS}(x_i, y_i))$
20: $E = \lambda_2 \|\hat{y}_i - w_{ri}\|_1$;
21: $\Delta \boldsymbol{W}_g \leftarrow \Delta \boldsymbol{W}_g + \frac{\partial \mathcal{L}_g}{\partial \boldsymbol{W}_g} + \frac{\partial \mathcal{L}_{g_{DLS}}}{\partial \boldsymbol{W}_g} + \frac{\partial E}{\partial \boldsymbol{W}_g}$;
22: $\Delta \boldsymbol{W}_d \leftarrow \Delta \boldsymbol{W}_d + \frac{\partial \mathcal{L}_d}{\partial \boldsymbol{W}_d} + \frac{\partial \mathcal{L}_{d_{DLS}}}{\partial \boldsymbol{W}_d}$;
23: **end for**
24: $\boldsymbol{W}_g \leftarrow \boldsymbol{W}_g - \frac{lr}{b} \Delta \boldsymbol{W}_g$;
25: $\boldsymbol{W}_d \leftarrow \boldsymbol{W}_d - \frac{lr}{b} \Delta \boldsymbol{W}_d$;
26: **end for**

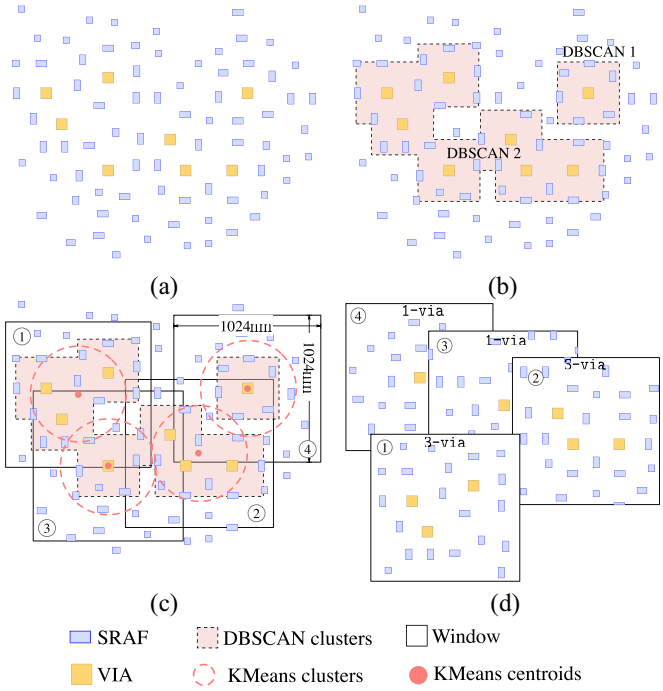

Fig. 6. Two-step full-chip splitting algorithm. (a) Part of full chip. (b) Coarse step: full chip to DBSCAN clusters. (c) Fine step: run KMeans++ on each DBSCAN cluster to get KMeans clusters, where each KMeans cluster belongs to a 1024×1024 nm^2 window. (d) Split chips.

design ($\boldsymbol{w}_r$) thus optimizing the mask directly. DAMO controls the whole flow from design to wafer while GAN-OPC relies on the conventional ILT engines. The training process of DAMO is described as Algorithm 1. G_{DMG}, D_{DMG}, and G_{DLS} are parameterized as $\boldsymbol{W}_g$, $\boldsymbol{W}_d$, and $\boldsymbol{W}_{g_{\text{DLS}}}$, respectively. First, we sample a mini-batch of design/mask/wafer patterns as ground truth and initialize $\mathcal{Z}_0/\mathcal{Z}$ for mask and wafer prediction, respectively (lines 3–7). Then, designs $\mathcal{W}$ and $\mathcal{Z}_0$ are concatenated for mask generation (line 8). Initially, the gradients of DMG are set to zeros (line 9). DMG is trained and the outputs are pushed into DLS for wafer prediction (lines 10–19). An L_1 loss E is applied to narrow the difference between ground-truth design $\boldsymbol{w}_{ri}$ and predicted wafer pattern $\hat{y}_i$ (line 20). In the training process, the parameters of G_{DMG} are not only updated by $(\partial \mathcal{L}_g / \partial \boldsymbol{W}_g)$ but also guided by $(\partial \mathcal{L}_{g_{\text{DLS}}} / \partial \boldsymbol{W}_g)$ (line 21). Similarly, D_{DMG} is optimized by $\mathcal{L}_d$ and $\mathcal{L}_{d_{\text{DLS}}}$ (line 22). Finally, the network gradients are calculated (lines 24 and 25).

With the guidance of DLS, the DAMO framework has a higher solution space than GAN-OPC. The success of our approach is also verified by various experiments. Compared to previous works, there are several advantages of DAMO as follows.

1) DLS surpasses LithoGAN [14] by being able to predict lithography contours of a single clip with multiple via patterns, which enables efficient training of DMG.
2) DAMO, equipped with DCGAN-HD, can directly output manufacturing friendly masks that avoid further fine-tuning with traditional costly OPC engines.

V. Full-Chip Splitting Algorithm

DAMO shows advantages on 1024×1024 nm^2 clips. To further adopt DAMO on full-chip layouts, a coarse-to-fine window splitting algorithm is proposed, in which the two-step clustering enables us to deal with full-chip industrial layouts where via patterns are distributed randomly with different local densities. Further, graph-based computation and parallelism techniques are used to accelerate the computations.

A. Two-Steps Splitting

A portion of one full-chip is shown in Fig. 6(a). First, we cluster the vias coarsely with DBSCAN. Second, the clustered vias are assigned to fine clusters by using KMeans++.

Coarse Step (DBSCAN): The main concept of the DBSCAN algorithm is to locate the regions of high via density that are separated from other low density regions. The distances between all pairs of vias are computed. For each via, if the number of neighboring vias is greater than a given threshold `minPts`, this via will be regarded as a core via. Any via neighborhood within a circle of radius Eps(ϵ) from a core via v will be assigned to the same cluster of v. Then, DBSCAN iteratively parses all of the core vias to determine all of the clusters. The DBSCAN algorithm is used to initially detect the clusters of via patterns (lines 1–4 in Algorithm 2). After the coarse step, the via patterns in a large layout will be assigned into different DBSCAN clusters, as shown in Fig. 6(b). Since the DBSCAN algorithm only needs two parameters: 1) the `minPts` and 2) the radius Eps(ϵ), while the size of the layout is not necessary for DBSCAN. This feature allows us to obtain

Algorithm 2 Full-Chip Splitting Algorithm

Require: Full-chip, DBSCAN parameter ϵ;
Ensure: Best full-chip splitting windows;
1: $\mathcal{V} \leftarrow$ collection of all via patterns; ▷ DBSCAN starts.
2: `minPts` $\leftarrow$ 1;
3: Run DBSCAN on $\mathcal{V}$ with parameters ϵ and *MinPts*;
4: $\mathcal{D} \leftarrow$ collection of DBSCAN clusters. ▷ DBSCAN ends.
5: $\mathcal{S} \leftarrow$ empty collection of best splitting windows;
6: $K \leftarrow$ max via number in a window; ▷ KMeans++ starts.
7: $H \leftarrow$ width and height of a window;
8: **for** each $d \in \mathcal{D}$ **do**
9: $V \leftarrow$ via number in DBSCAN cluster d;
10: **for** $\forall k < V$ **do**
11: Run KMeans++ in cluster d with k centroids;
12: $\mathcal{C} \leftarrow$ collection of KMeans clusters in DBSCAN cluster d;
13: Create $H \times Hnm^2$ split windows centered at k centroids;
14: *BestSplitting* $\leftarrow$ True;
15: **for** each KMeans cluster $c \in \mathcal{C}$ **do**
16: $v_c \leftarrow$ via number of KMeans cluster c;
17: **if** $v_c > K$ or via in c is not in k split windows **then**
18: *BestSplitting* $\leftarrow$ False;
19: Break;
20: **end if**
21: **end for**
22: **if** *BestSplitting* is True **then**
23: Add the k split windows to $\mathcal{S}$;
24: **end if**
25: **end for**
26: **end for**
27: **return** collection of best splitting windows $\mathcal{S}$; ▷ KMeans++ ends.

a density-based preliminary clustering result for a layout with an arbitrary size.

Fine Step (KMeans++): After DBSCAN clustering, every via pattern is assigned to a coarse cluster d which contains V via patterns. Then, we search every coarse cluster and run KMeans++ algorithm to find the best splitting windows, where the max number of via patterns in a window is set to K (lines 5–27 in Algorithm 2). Note that every KMeans cluster belongs to a 1024×1024 nm^2 window, whose center locates at the centroid of the KMeans cluster, as shown in Fig. 6(c).

After the coarse-to-fine clustering, the design will be split into many 1024×1024 nm^2 windows [see Fig. 6(d)]. Our coarse-to-fine splitting algorithm has many advantages. First, it is extremely fast because DBSCAN only needs to scan the via patterns once, and it also skips the empty areas. Second, the typical window-sliding method is hard to handle overlapping situations and stitching errors. We conduct a clustering algorithm to make sure each via belongs to one and only one cluster. The splitting process is based on the clusters instead of the windows. Fig. 7(a) gives one example. We mark two patterns via-A and via-B with red borders. The via-A is in both window-1 and window-3, while it only belongs to KMeans cluster C1. According to the clustering result, the via-A is too far from the centroid of cluster C3 to influence the mask optimization process in cluster C3. Therefore, the algorithm only assigns the via-A to the window-1 to form the final chip. Thus, overlapping situations and stitching errors will not occur. The splitting result can be visualized hierarchically in Fig. 7(b).

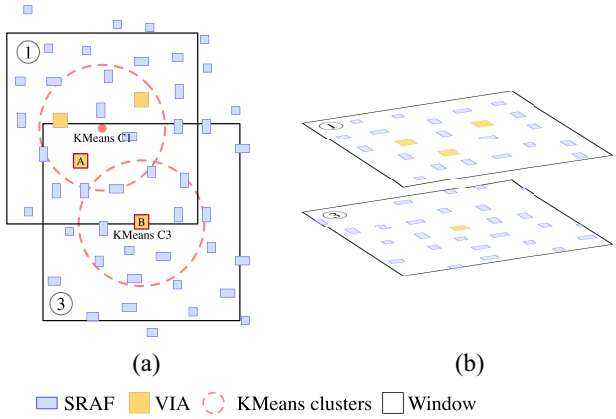

Fig. 7. Detail illustration of the KMeans clusters and the chips. (a) KMeans++ clusters, we illustrate only cluster 1 and cluster 3 for simplicity. (b) Hierarchical visualization for chip 1 and chip 3.

Note that in this article we focus on the mask optimization process, while all the SRAFs are generated directly by Mentor Calibre. During the mask optimization, we only replace the design patterns with our optimized masks. The SRAFs remain unchanged in the final large chip.

Third, because the window locates at the centroids of the clusters, the via patterns are all placed near the center of the windows, which reduces the search space of the machine learning model to a large extent, resulting in less training data and training time.

B. Acceleration Techniques on GPU

Typically, the DNN models, including our proposed DCGAN-HD, can be deployed on GPU to accelerate the inferences. In contrast, traditionally, DBSCAN and KMeans++ are implemented on the CPU. The computations on the CPU are conducted via fetch–decode–execute cycles, in which the instructions are fetched, decoded, and executed one by one, thus very slow. Computations on CPU also usually suffer from cache misses due to the limited sizes of on-chip caches. Parallel computing on CPU has been proposed while the scheduling is actually pseudoparallel therefore the deployment performance is usually unsatisfying. In comparison, GPU facilitates parallel computations by following the single-instruction–multiple-data (SIMD) mechanism. On GPU, there are a group of blocks, and in each block, there are some threads that can compute on different data following the same instruction. The blocks and threads share some global buffers and also own some local buffers exclusively. These blocks and threads can conduct the same operations on different data and then merge them or communicate with each other. In other words, a task can be divided hierarchically to be operated cooperatively by the blocks and threads in parallel. For the clustering algorithms used in this article, the irregular computation and communication patterns hinder the performance improvements on the CPU. Consequently, there is a significant difference in speed between Algorithm 2 on CPU and DCGAN-HD on GPU, and Algorithm 2 has been a bottleneck of our flow. To address these issues, graph-based computation and parallelism

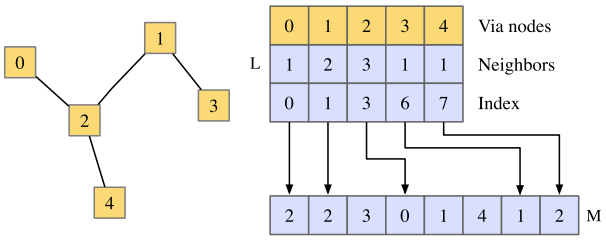

Fig. 8. Example of the graph representation for the on-chip vias, and the corresponding lists. The number of vias is $V = 5$, and the number of edges is $E = 4$.

techniques are adopted to accelerate the computations of DBSCAN and KMeans++ on GPU.

As mentioned above, the operation of DBSCAN is based on calculating a distance between each pair of objects, which is defined according to the adopted metric. The objects are grouped with each other if their distance is smaller than the given radius Eps(ϵ). The complexity of DBSCAN is $\mathcal{O}(n^2)$, thus making its scalability a major challenge while tackling large data volume scenarios.

In this work, we propose to represent the vias to be clustered as a graph with an indexing matrix and a neighbor list [31]. Then, the clustering process is implemented for each vertex in this graph in parallel on GPU. Represent the vias on the chip as a vertex set $\mathcal{V}$. First, computing the distances between each pair of vias and connecting two vias if their distance is smaller than the radius Eps(ϵ). Consequently, we can obtain a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is the edge set. While computing $\mathcal{E}$, each via is assigned to a GPU thread. Current GPUs usually support the parallel computations of more than thousands of threads at the same time. Therefore, this process is much faster compared with the serial computations on the CPU. Fig. 8 is taken as an example to illustrate the graph representation. The vias are stored in an indexing matrix M with two rows. The first row records the number of neighboring vias for each via at the location of its index. The second row stores the starting position of the neighborhood in the neighbor list. The edge information is stored as a neighbor list L, where the starting position for each via is stored in M. While calculating the pairwise distances, and determine that whether a via is a core, the complexity can be reduced by assigning threads to each via, and share the graph data between the vias. Note that there exists data dependency while computing the starting indices of the neighborhood lists, i.e., the successor vias depend on processor vias. This problem can be tackled easily by using exclusive scan operation on GPU [32].

After the creation of graph representation, the clustering process is done by using the breadth-first search (BFS) method. A group of threads is created, and each thread conducts a BFS procedure starting at each core via. The BFS will visit all of the neighboring vias in L. As the vias are visited, they are labeled as members of the same cluster. Each BFS will find a new cluster and will be repeated as long as there are nonvisited core vias. Denote the number of vias as V, and the number of edges as E. Theoretically, the graph construction has the time complexity $\mathcal{O}(V^2)$ since it needs to compute the distance between each pair of vias. In practice, some pairs will not be computed since they are far between each other. The BFS clustering stage has the complexity $\mathcal{O}(V + E)$. Therefore, the theoretical complexity is $\mathcal{O}(V^2)$ in the worst case. However, the parallelism on GPU reduces the time complexity significantly.

The drawback of the KMeans++ is that to find the optimal centroids, in each optimization iteration, pairwise distances between cluster centroids and noncentroids vias are required. Sequentially computing the k centroids makes the process very slow on CPUs for even a very small k. Given a set of selected centers C, the objective is to minimize the errors between the vias in $\mathcal{V}$ and C. Initially, a center via v_0 is sampled uniformly at random and then $k - 1$ cluster centers are sampled adaptively according to the D^2-sampling method. Specifically, in iteration $i \in \{2, \ldots, k\}$, the via $v \in \mathcal{V}$ is sampled to the set of already sampled cluster centers C_{i-1} with probability

$$p(v) = \frac{\mathrm{d}(v, C_{i-1})^2}{\sum_{v' \in \mathcal{V}} \mathrm{d}(v', C_{i-1})^2} \tag{13}$$

where $\mathrm{d}(\cdot, \cdot)$ is the distance function. The computational complexity is $\Theta(nk)$, arising from the computations of the distances between pairs of vias. n is the number of vias. In our applications, there are dozens of vias (i.e., n) in each cluster after clustering via DBSCAN, and k is usually set to be 5. Consequently, the computations can be paralleled on GPU elegantly, since the number of available threads on GPU is usually larger than the number of vias.

VI. Experimental Results

Many experiments are carried out to evaluate our proposed framework. First, we evaluate the effectiveness of our DLS by testing the mIoU and pixAcc of generated wafer patterns. Second, the superiority of our proposed DAMO is also validated by thorough experiments. Finally, we test our model using the full-chip layout in ISPD 2019 contest [33], which is generated by an open-source router [34].

A. Dataset

Our Training Set and Validation Set: As described in Section IV-A, two sets of 2048×2048 pixels RGB images are generated for training purposes: one mask–wafer paired for DLS, while another one design–mask–wafer paired for DMG. To obtain fine-grained models, we divide our data depending on the via number with a window, and six groups marked as `1-via`, `2-via`, ..., `6-via` are generated. For instance, the `1-via` group contains all cases with only one via in a window. Each group has 2000 training images and 500 validation images.

ISPD 2019 Large Full-Chip Test Set: We use another real benchmark coming from ISPD 2019 Contest on Initial Detailed Routing, We take the layer 40 of `ispd19_test1` [33] as our design layer (100×100 um). After the SRAF insertion, OPC, and lithography process via Calibre, we extract the design, SRAF, mask, and wafer layers and merge them to be the ground truth. Then, using our coarse-to-fine full-chip splitting algorithm, the full-chip layout

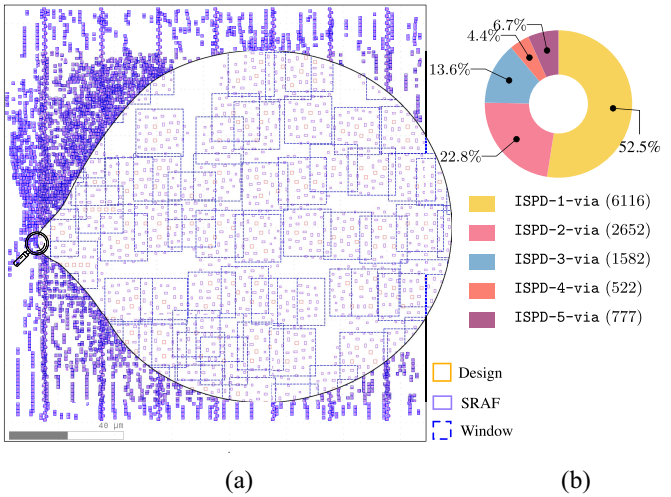

Fig. 9. (a) ISPD 2019 large full-chip layout and splitting windows. (b) Via window distribution in `ispd19_test1` [33].

is split to lots of 1024×1024 nm^2 layout windows. According to the design rule, we set the DBSCAN radius Eps (ϵ) to be 400 nm. The hyperparameters K in KMeans++ fine step is set to 5, because the images containing more than five design patterns only account for 0.5% in the total windows. The `ispd19_test1` benchmark contains 16 035 design patterns which are split to 11 649 windows. 6116 split windows marked as `ISPD-1-via` has only one via in a window, accounting for 52.5%. The detailed distribution of different windows is illustrated in Fig. 9.

B. Implementation Details

The proposed DAMO is implemented in Python with PyTorch library [35]. In both DLS and DMG training stages, The Adam optimizer [36] is adopted, where we set base learning rate and momentum parameters to 0.0002 and (0.5, 0.999). In the LeakyReLU, the slope of the leak is set to 0.2 in all models. We set the batch size to be 4, and the maximum training epoch to be 100. The weight parameters of λ_0, λ_1, α_0, α_1, and λ_2 are set to 100, 100, 30, 30, and 20, respectively. The fixed parameters of DLS can be implemented by the evaluation mode of PyTorch. After training, the generated mask layer will be converted into a GDSII layout file then fed into Mentor Calibre for lithography simulation validation. We use four Nvidia TITAN Xp GPUs for training and one for testing. The evaluation metrics we adopt are mIoU, pixAcc, L_2 error, and PV Band. Here, the PV Band is calculated by Calibre.

C. Effectiveness of DLS

Before training DAMO, it is of great importance to construct a high-performance DLS. Since our DLS model is based on the cGAN framework, we set up an ablation experiment to illustrate the advantages of our generator and discriminators. The results in Table II are the average values from six validation sets. First, cGAN (used in LithoGAN) provides a baseline mIoU of 94.16% which is far away from practical application. Then, UNet++ is used to replace the UNet generator in cGAN for better performance. However, the original UNet++ is not qualified to be a generator of a cGAN and the mIoU is reduced to 93.98%.

TABLE II
RESULTS OF DLS

Generator	Discriminator	Loss	mIoU (%)	pixAcc (%)
UNet	D (cGAN)	L_1	94.16	97.12
UNet++	D (cGAN)	L_1	93.98	96.74
G (Our)	D (cGAN)	L_1	96.23	97.50
G (Our)	D (Our)	L_1	97.63	98.76
G (Our)	D (Our)	L_p	98.68	99.50
G (Our)	D (Our)	L_{MSSIM}	98.24	99.01
G (Our)	D (Our)	$L_p + L_{MSSIM}$	**98.74**	**99.57**

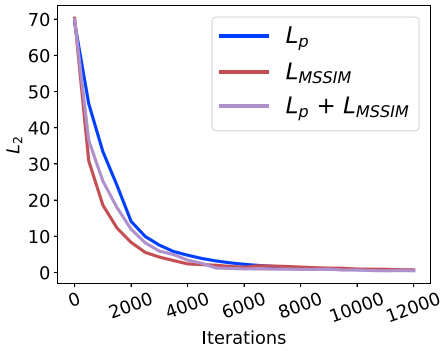

Fig. 10. Visualization of L_2 with different loss function during the training process.

Following DCGAN, we made some amendments in UNet++ (as discussed in Section III-B1) and high-resolution generator is adopted in our DLS model. After applying our high-resolution generator, mIoU is improved to 97.63%, which outperforms UNet and UNet++ generators by a large margin when using the same discriminator. The huge gain in mIoU implies that our developed high-resolution generator is a strong candidate for DLS. Next, the newly designed multiscale discriminators (introduced in Section III-B2) are used to replace the original cGAN discriminator. Results in Table II show that mIoU is further boosted to 97.63%.

We replace the L_1 loss with the perceptual loss proposed in Section III-B3 and the mIoU reaches 98.68%. In addition, we design a set of ablation studies on the SSIM loss and perceptual loss. Compared with the perceptual loss, the mIoU and the pixAcc of SSIM loss are reduced by 0.44% and 0.49%, respectively. As shown in Fig. 10, the SSIM loss can achieve faster convergence than the perceptual loss. In the EDAMO framework, we combine the SSIM loss with the perceptual loss to further improve mIoU to 98.74% and pixAcc to 99.57%.

Additionally, DLS can handle multiple vias in a single clip, which overcomes the limitation of LithoGAN [14].

D. Performance of DAMO

We test DAMO on the six groups of validation sets to verify the performance. Every generated mask will be pushed into Calibre for lithography simulation. After that, we apply L_2 and PV Band measurements to test the performance of different

TABLE III
COMPARISON WITH STATE OF THE ART ON VALIDATION SET

Bench	case#	Robust-OPC [4]			GAN-OPC [9]			Calibre			DAMO [37]			Enhanced DAMO		
		L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)
1-via	500	1510	3198	3865	1464	3064	321	1084	2918	1417	1080	2917	284	1075	2909	203
2-via	500	4704	7070	3779	4447	6964	336	2161	5595	1406	2129	5576	281	2121	5562	199
3-via	500	8402	11619	3907	8171	11426	317	3350	8286	1435	3244	8271	285	3207	8254	205
4-via	500	12035	15315	3962	11659	14958	327	4331	10975	1477	4263	10946	291	4225	10917	204
5-via	500	16319	19136	3796	15773	18976	318	5410	13663	1423	5396	13640	279	5344	13611	198
6-via	500	19671	23004	3840	18904	22371	320	6647	15572	1419	5981	15543	284	5913	15497	201
Average		10440	13223	3858	10069	12960	323	3831	9502	1430	3682	9482	284	**3647**	**9458**	**201.6**
Ratio		2.835	1.394	13.585	2.735	1.367	1.138	1.040	1.002	4.427	1.000	1.000	1.000	0.991	0.997	0.710

TABLE IV
COMPARISON ON ISPD 2019 FULL-CHIP SPLITTING WINDOWS

Bench	case#	Robust-OPC [4]			GAN-OPC [9]			Calibre			DAMO [37]			Enhanced DAMO		
		L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)	L_2	PVB	T (s)
ISPD-1-via	6116	2519	3603	34390	2367	3492	3963	1073	2857	18959	1056	2848	3669	1049	2841	2688
ISPD-2-via	2652	5701	7418	19129	5412	7126	1742	2232	5670	7537	2172	5654	1591	2149	5617	1151
ISPD-3-via	1582	8979	13799	11575	8792	13047	1021	3602	8276	4494	3196	8127	949	3110	8067	694
ISPD-4-via	522	12756	15834	3840	12395	15015	341	4395	11051	1692	4361	10987	313	4298	10817	228
ISPD-5-via	777	17010	20107	4317	16526	19147	495	5526	12305	2537	4542	12251	466	4465	12097	335
Average		9393	12152	14650	9098	11565	1512	3365	8031	7043	3065	7973	1397	**3014**	**7887**	**1019**
Ratio		3.064	1.524	10.487	2.968	1.451	1.082	1.098	1.007	5.041	1.00	1.00	1.00	0.983	0.989	**0.73**

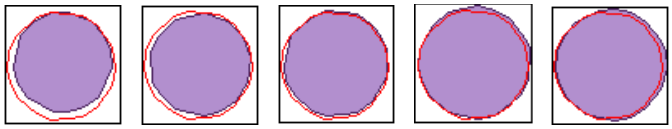

Fig. 11. Visualization of DAMO model advancement on via layer: (a) Epoch 20, (b) Epoch 40, (c) Epoch 60, (d) Epoch 80, and (e) Epoch 100.

mask optimization methods. Note that since GAN-OPC fails to train on high-resolution input, the 1024×1024 input images are downsampled to 256×256 pixels to train the model. After the inference process, the results are upsampled to the original size for L_2 and PV Band testing. Table III shows that on the validation set, DAMO has 2.7× less L_2 error and 1.3× less PV Band compared with GAN-OPC. In addition, DAMO outperforms Calibre in both L_2 and PV Band metrics, meanwhile achieving 4× speed-up. The L_2, the PV Band, and the runtime performance of DAMO are better than Calibre and GAN-OPC in all cases, which demonstrates that the stability of DAMO can be guaranteed.

The mask optimization process of DAMO is visualized in Fig. 11. All the wafer images are generated using Calibre lithography simulation. The red contours represent wafer patterns on masks produced by Calibre while the purple wafers are on masks generated by DAMO. We sample DAMO results after 20/40/60/80/100 training epochs for the illustration. Initially, the wafer patterns of DAMO have lower quality compared with Calibre [as shown in Fig. 11(a) and (b)]. Along with the increase of training epochs, the results of DAMO and Calibre are getting closer [Fig. 11(c)–(e)] show that the performance of DAMO surpasses Calibre after the iterative optimization.

E. Results on ISPD 2019 Full-Chip Layout

For ISPD 2019 large full-chip layout, the experiment has two stages. In the first stage, we test DAMO on the 11 649 split windows, as listed in Table IV. We compare GAN-OPC, Calibre, and DAMO under metrics of L_2, PV Band, and runtime. Here, columns "L_2," "PVB," and "T (s)" represent L_2 (nm^2), PV Band (nm^2) and runtime (s). DAMO shows better performance against Calibre and GAN-OPC, on all metrics of L_2, PV Band, and runtime. Comparing the results listed in Tables III and IV, since the simple patterns, such as ISPD-1-via and ISPD-2-via, account for a great proportion (75.3%), the average of L_2 or PV Band performs better in the ISPD 2019 dataset than our validation set.

In the second stage, we recover all the split windows into the original 100×100 um^2 large full-chip layout with DAMO generated masks. Still, we use Calibre to test the L_2 error and PV Band of the large layout results. Fig. 12 shows the sum of L_2 error and PV band on split windows are very close to

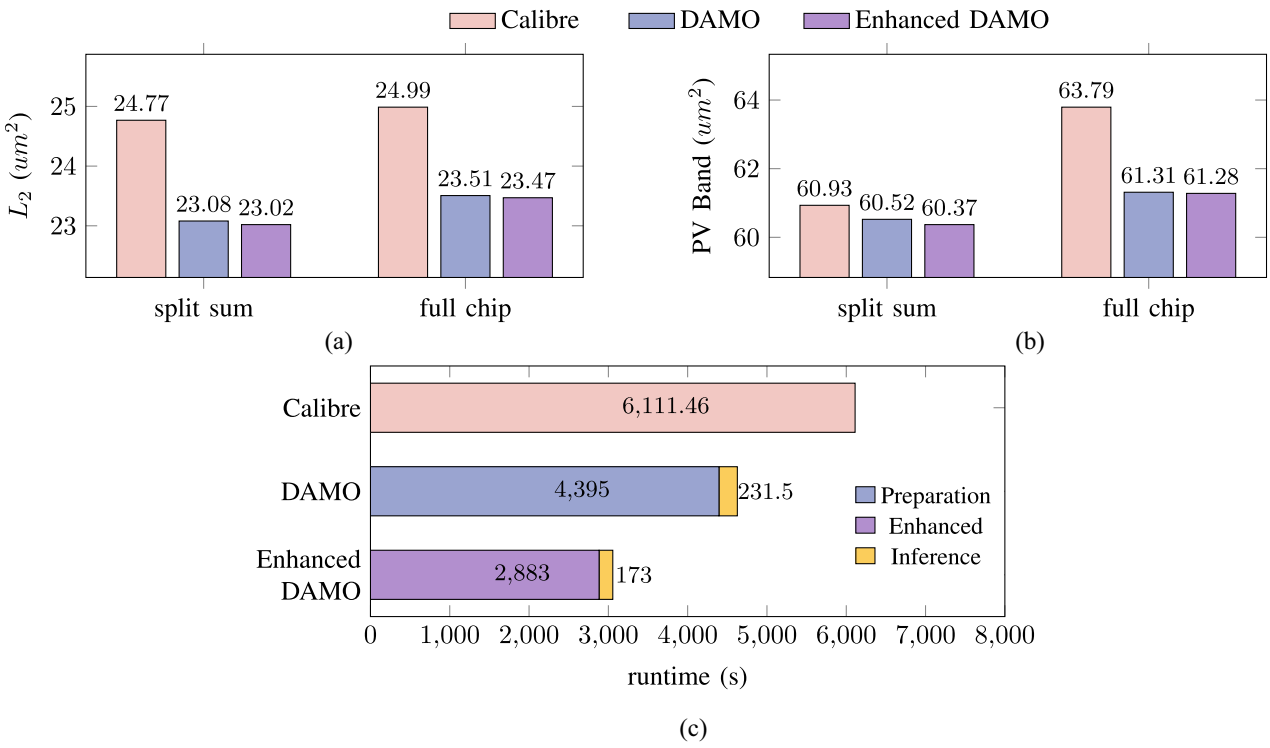

Fig. 12. Comparison with Calibre on ISPD 2019 full-chip layout in terms of (a) L_2, (b) PV Band, and (c) runtime.

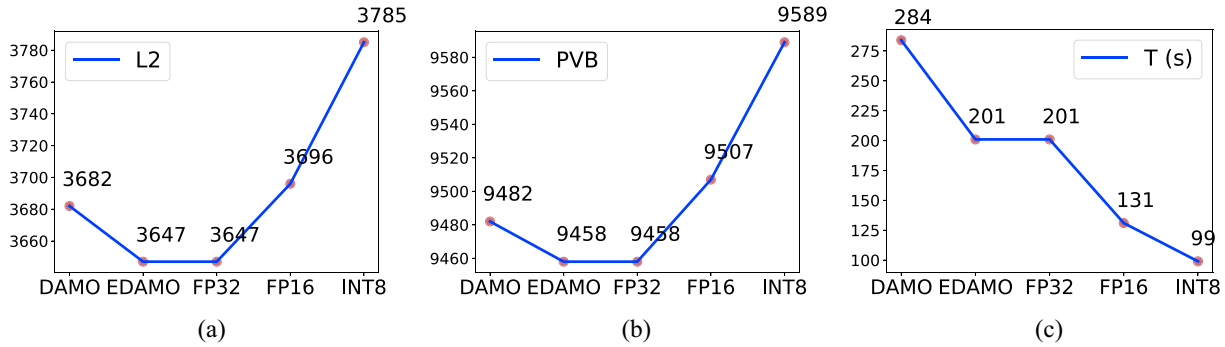

Fig. 13. Comparison among the original DAMO, the EDAMO, and TensorRT FP32, FP16, INT8 precisions.

the results of full-chip layouts owing to our efficient splitting algorithm. As shown in Fig. 12(a) and (b), DAMO still has better performance than Calibre. For the runtime of the large full-chip layout [see Fig. 12(c)], we separate runtime of DAMO to preparation time (4395 s) and inference time (231.5 s). The inference time takes only 5% of the total by parallel using four GPUs. Preparation includes the full-chip splitting, split layouts to images, generated images to layouts, and the split windows to full-chip recovering. All these preparation processes are running on a single CPU, which means the preparation time can be easily reduced when using multi CPUs in parallel.

F. Performance of the Enhanced DAMO

Our EDAMO has been implemented on NVIDIA GPUs to accelerate the computations, and the data precision is 32-bit floating point (FP32). The deep learning layers are deployed via NVIDIA TensorRT [38], and the full-chip splitting algorithm is implemented as CUDA C++ on GPUs. TensorRT is designed to facilitate the high-performance computations of deep learning models on NVIDIA GPUs and focuses on accelerating the inference of an already-trained model. Some typical optimization techniques provided by TensorRT include layer fusions (i.e., combining neighboring layers into one computation core to reduce the interlayer communications), kernel selections (i.e., selecting optimal CUDA kernels for the operations in the DNN model), conversions of operations (i.e., converting operations to high-performance matrix operations which are suitable for GPUs), etc. The convolutions, deconvolutions, and residuals in our DAMO are perfectly supported by TensorRT, which supports mixed-precision inference with FP32, FP16, or INT8. As shown in Fig. 13, we test the EDAMO with those three precisions. When using FP32, the accuracies of EDAMO with SSIM loss are better than original DAMO, while FP16 and INT8 will cause accuracy losses. Here, we apply mixed-precision

training to reduce the accuracy gap between FP32 and FP16. However, our proposed splitting algorithms are not supported by TensorRT. To avoid the unnecessary communication and synchronization costs between CPU and GPU, Algorithm 2 is also implemented on GPU by ourselves, so as to make the split chips passed into DAMO on GPU seamlessly.

The experimental results are shown in Tables III and IV and Fig. 12. Columns "L_2," "PVB," and "T (s)" represent L_2 (nm^2), PV Band (nm^2), and runtime (s). On the validation set, the average reduction of the running time is reduced by up to 29%. On the ISPD dataset, the average reduction is 27%. We also implement a simple version of Robust-OPC [4]. Since the lithography model in this article is different from that in Robust-OPC [4], the runtime performance of Robust-OPC listed in this article is severely compromised due to the data saving and loading with GDSII format. As shown in Table III, with SSIM loss, the L_2 and PV Band of the EDAMO reduce 0.9% and 0.3%, respectively, on the validation set. Similarly, in Table IV, the L_2 and PV Band also reduce 1.7% and 1.1%. On the ISPD 2019 full-chip dataset, the EDAMO also outperforms the DAMO framework, detailed in Fig. 12(a) and (b). In Fig. 12(c), the profiling of the running times is plotted. The running time of data preparation (including full-chip splitting) reduces from 4395 to 2883 s in the GPU-accelerated version. The model inference part reduces from 231.5 to 173 s with the help of TensorRT. Overall, the improvements of our EDAMO are 34.01% and 50.04% compared with the traditional DAMO and Calibre, respectively.

VII. Conclusion

In this article, we present DAMO, an end-to-end framework targeting full-chip mask optimization with high-resolution generative machine learning models. The framework comes with DLS that offers precise lithography prediction benefiting from the proposed DCGAN-HD. The high-quality DLS also enables the efficient training of DMG which hence promises to generate manufacturing friendly masks without further costly fine-tuning. The advantage of the proposed framework over the representative industrial and academic state of the art demonstrates the possibility of deep neural networks as an alternative solution to many layout and mask optimization problems. Our future research includes the deployment of the framework to more complicated designs (such as metal layers) and the transferability as technology node advances.

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] H. Yang *et al.,* "VLSI mask optimization: From shallow to deep learning," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, Beijing, China, 2020, pp. 434–439.

[3] 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 1st Int. Symp. Qual. Electron. Design*, San Jose, CA, USA, 2000, pp. 81–85.

[4] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *Proc. IEEE/ACM Design Autom. Test Eur. (DATE)*, Grenoble, France, 2015, pp. 1591–1594.

[5] 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.*, vol. 35, no. 8, pp. 1345–1357, Aug. 2016.

[6] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," *J. Micro Nanolithogr. MEMS MOEMS*, vol. 15, no. 2, 2016, Art. no. 021009.

[7] 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 Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2014, pp. 1–6.

[8] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Irvine, CA, USA, 2017, pp. 81–88.

[9] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Y. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. 55th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2018, pp. 1–6.

[10] B. Jiang, H. Zhang, J. Yang, and E. F. Y. 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.*, vol. 39, no. 10, pp. 2849–2859, Oct. 2020.

[12] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2020, pp. 1–6.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2016, pp. 785–794.

[14] 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 Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[15] L. Pang *et al.*, "Study of mask and wafer co-design that utilizes a new extreme SIMD approach to computing in memory manufacturing: Full-chip curvilinear ILT in a day," in *Proc. Photomask Technol.*, vol. 11148. Monterey, CL, USA, 2019, Art. no. 111480U, doi: 10.1117/12.2534629.

[16] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[17] *Calibre Verification User's Manual*, Mentor Graph., Wilsonville, OR, USA, 2008.

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 1125–1134.

[19] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014. [Online]. Available: arXiv:1411.1784.

[20] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 2672–2680.

[21] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Y. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2822–2834, Oct. 2020.

[22] Q. Chen and V. Koltun, "Photographic image synthesis with cascaded refinement networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1520–1529.

[23] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional GANs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 8798–8807.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervention*, 2015, pp. 234–241.

[25] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "UNet++: A nested U-Net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Cham, Switzerland: Springer, 2018, pp. 3–11.

[26] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 694–711.

[27] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," 2015. [Online]. Available: arXiv:1505.00853.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.
[29] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, pp. 600–612, 2004.
[30] H. Yang, W. Chen, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Automatic layout generation with applications in machine learning engine evaluation," 2019. [Online]. Available: arXiv:1912.05796.
[31] G. Andrade, G. Ramos, D. Madeira, R. Sachetto, R. Ferreira, and L. Rocha, "G-DBSCAN: A GPU accelerated algorithm for density-based clustering," *Procedia Comput. Sci.*, vol. 18, pp. 369–378, Jan. 2013.
[32] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA," *GPU Gems*, vol. 3, no. 39, pp. 851–876, 2007.
[33] (2018). *ISPD 2019 Contest on Initial Detailed Routing*. [Online]. Available: http://www.ispd.cc/contests/19/#benchmarks
[34] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young, "Dr. CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Westminster, CO, USA, 2019, pp. 1–7.
[35] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Workshop*, 2017, pp. 1–4.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.
[37] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. 39th Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.
[38] *NVIDIA TensorRT*. Accessed: Oct. 2021. [Online]. Available: https://developer.nvidia.com/tensorrt

Guojin Chen received the B.Eng. degree in software engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2019. 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 deep learning in VLSI design for manufacturability, high-performance computing, and computer vision.

Wanli Chen received the B.Eng. degree from the Southern University of Science and Technology, Shenzhen, China, in 2018. 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 high-performance computer vision and machine learning.

Qi Sun (Graduate Student Member, IEEE) received the B.Eng. degree in computer science from Xidian University, Xi'an, China, in 2018. 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 deep neural network hardware acceleration, high-level synthesis, and design space exploration.

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 has interned with Cadence Design Systems, San Jose, CA, USA; NVIDIA Research, Austin, TX, USA; and Tencent Youtu X-Lab, Shenzhen, China. His research interests include VLSI design for manufacturing, physical design, and machine learning on chips.

Dr. Ma received the Best Paper Award from ASPDAC 2021, the Best Student Paper Award from ICTAI 2019, the Best Paper Award Nomination from ASPDAC 2019, and the Best Poster Research Award from Student Research Forum of ASPDAC 2020.

Haoyu Yang received the B.E. degree from Qiushi Honors College, Tianjin University, Tianjin, China, in 2015, 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 a Research Scientist with NVIDIA ASIC, Austin, TX, USA, and VLSI Research Group, Austin. His research interests include machine learning in VLSI design for manufacturability, high-performance VLSI physical design with parallel computing, and machine learning security.

Dr. Yang received the Nick Cobb Scholarship from SPIE 2019 Advanced Lithography, the Best Paper Award nomination from ASPDAC 2019, and the Best Poster Presentation from Student Research Forum of ASPDAC 2019.

Bei Yu (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 seven Best Paper Awards from ASPDAC 2021, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, ICCAD 2013, ASPDAC 2012, and six ICCAD/ISPD contest awards. He has served as the TPC Chair of 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.