Neural-ILT 2.0: Migrating ILT to Domain-Specific and Multitask-Enabled Neural Network

Bentian Jiang, Lixin Liu, Yuzhe Ma, *Member, IEEE*, Bei Yu, *Member, IEEE*, and Evangeline F. Y. Young, *Senior Member, IEEE*

***Abstract*—Optical proximity correction (OPC) in modern design closures has become extremely expensive and challenging. Conventional model-based OPC encounters performance degradation and large process variation, while aggressive approach, such as inverse lithography technology (ILT), suffers from large computational overhead for both mask optimization and mask writing processes. In this article, we developed Neural-ILT, an end-to-end learning-based OPC framework, which literally conducts mask prediction and ILT correction for a given layout in a single neural network, with the objectives of: 1) mask printability enhancement; 2) mask complexity optimization; and 3) flow acceleration. A domain-specific model pretraining recipe, which introduces the domain knowledge of lithography system, is proposed to help Neural-ILT achieving faster and better convergence. Quantitative results show that compared to the state-of-the-art (SOTA) learning-based OPC solutions and conventional OPC flows, Neural-ILT can achieve 15× to 30× turnaround time (TAT) speedup and the best mask printability with relatively lower mask complexity. Based on the developed infrastructure, we further investigated the feasibility of handling multiple mask optimization tasks for different datasets within a common Neural-ILT platform. We believe this work could bridge well-developed deep learning toolkits to GPU-based high-performance lithographic computations to achieve groundbreaking performance boosting on various computational lithography-related tasks.**

***Index Terms*—Computational lithography, deep neural networks, inverse lithography technique (ILT), optical proximity correction (OPC).**

I. Introduction

COMPUTATIONAL lithography models are designed to mimic the printing effects of real lithography patterns. Building on top of these delicate lithographic models, resolution enhancement techniques (RETs), such as subresolution assist feature (SRAF) insertion and optical proximity correction (OPC), help the designers to obtain optimized masks that result in high fidelity printed patterns.

Manuscript received 13 March 2021; revised 16 June 2021; accepted 14 August 2021. Date of publication 1 September 2021; date of current version 19 July 2022. This work was supported in part by the Research Grants Council of the Hong Kong SAR under Project CUHK14209320. The preliminary version has been presented at IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2020. This article was recommended by Associate Editor L. Behjat. *(Corresponding author: Bentian Jiang.)*

The authors are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: btjiang@cse.cuhk.edu.hk).

Digital Object Identifier 10.1109/TCAD.2021.3109556

As one of the most prevailing RETs, OPC modifies on-mask geometries under the guidance of a computational lithography simulator, which helps to counteract the effects of diffraction-related blurring and under-exposure issues for the given masks. Mainstream OPC approaches can be roughly categorized into: 1) model-based OPC [1], [2] and 2) inverse lithography technology-based (ILT) OPC [3]–[6]. Model-based OPC usually relies on compact model simulation to drive the movements of polygon edges, which are typically divided into segments for the reasons of mask manufacturability and computational efficiency, however, at the expense of limited solution space and performance bottlenecks. On the other hand, ILT uses numerical approach, which treats OPC as an inverse imaging problem, to perform pixelwise mask optimization. Nowadays, ILT is widely adopted to find flexible 193i and even EUV mask pattern solutions to improve overall process window [5], [7]. However, with continuous shrinkage of the technology nodes, the drastically rising of lithography computational overhead has brought great challenges for ILT to balance quality of results (QoR), speed, and affordability.

In the past decade, both academia and industry have been actively working on facilitating the conventional lithography-related processes as well as maintaining competitive QoR. Significant efforts have been made, including but not limited to: 1) migrating high-performance computational lithography to GPU acceleration [8]; 2) introducing fast modeling approaches for rigorous/compact litho-simulations [9]; 3) considering multiple patterning [10], [11]; and 4) applying the state-of-the-art (SOTA) machine learning techniques on lithography-related applications such as lithography system modeling [12], [13], hotspot detection [14]–[16], and OPC [12], [17]–[19]. Among them, Yang *et al.* [18] (GAN-OPC) for the first time applied conditional generative adversarial networks (CGANs) to mimic the process of typical ILT OPC. The predicted mask by GAN is treated as a better initial solution for a conventional CPU-based ILT tool [4] for further refinement, which helps to achieve faster convergence and better mask printability. Ye *et al.* [12] (LithoGAN) developed a CGAN-based lithography modeling framework that can directly map mask patterns to resist patterns, while achieving orders of magnitude speedup with acceptable accuracy loss.

Quantitative results of the above seminal works are encouraging, but they also reveal a crucial fact that, in most lithography-related application scenarios, machine learning is treated as a compromising solution to tradeoff between result quality and turn around time. For supervised learning, the

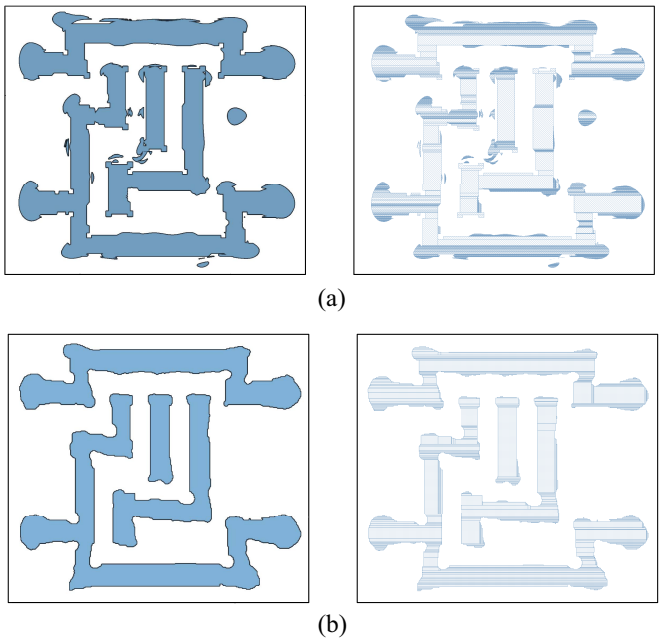


Fig. 1. Visualizations of mask fracturing results on the synthesized masks with different complexities. (a) Conventional ILT [4] synthesized mask (high complexity) and corresponding mask fracturing result with 1575 shots. (b) Neural-ILT synthesized mask (low complexity) and corresponding mask fracturing result with only 444 shots.

prediction quality is highly related to the quality of the training dataset, one may need to train different models for datasets with different distributions, and the inevitable prediction loss may further degrade its actual performance. From the perspective of realistic deployment, the prediction error in machine learning solutions usually requires additional rounds of rigorous refinements to ensure correctness, which weakens its practical value.

In addition, for OPC, the optimized masks need to be fractured as a combination of rectangular variable-shaped beam (VSB) shots for mask writing. The ideal curvilinear shapes [Fig. 1(a)] generated by conventional ILT [4] require huge amounts of shots to accurately replicate the shapes, which leads to extremely poor mask manufacturability due to the unmanageable mask writing times [7], [20]. Observing the fact that reducing the mask complexity can significantly reduce the shot count [2045 in Fig. 1(a) versus 653 in Fig. 1(b)], it is imperative to consider mask complexity as an objective during the ILT correction.

Motivated by these issues, a new challenge naturally arises: 1) can we completely replace the conventional ILT-based OPC flow by *purely* learning-based methods; 2) so as to simultaneously achieve breakthrough in runtime boosting and the SOTA result quality; and 3) while considering mask manufacturability? In this work, we develop Neural-ILT, an end-to-end high-performance ILT-based OPC framework based on a single neural network. Unlike previous learning-based OPC solutions [18], [19], a converged Neural-ILT is able to directly generate the masks after OPC for the unseen layouts and does not require any additional rigorous refinement on the network output. Our key contributions are summarized as follows.

1) We developed Neural-ILT, an end-to-end ILT correction framework based on a single neural network. A CUDA-based lithography simulation tool is developed and deeply embedded into Neural-ILT to enable on-neural-network lithographic computations.
2) A special training recipe with domain knowledge of a partial coherent lithography imaging system is applied to pretrain the backbone network of Neural-ILT, which helps to achieve faster and better convergence for the on-neural-network ILT correction.
3) The functionalities of conventional ILT correction and mask complexity refinement are cast as customized neural network layers and integrated into Neural-ILT. Consequently, the on-neural-network ILT correction is essentially the training procedure of Neural-ILT in an unsupervised manner.
4) We leveraged the existing Neural-ILT infrastructure to achieve rapid task adaptation from the domain of 32-nm metal-layer to the domain of 45-nm cut-layer, which verified the feasibility of multitask-enabled Neural-ILT paradigm.
5) Experimental results show that Neural-ILT achieves breakthrough turn around time speedup with lower mask complexity and the best mask printability.

The rest of this article is organized as follows. Section II lists some preliminaries. Section III discusses the details of the framework and algorithms. Section IV presents the experimental results, followed by a conclusion in Section V.

II. Preliminaries

In this section, we will go through the background and problem formulation. Throughout the rest of this article, we denote $\mathbf{Z}_{\text{t}}$ as the target layout, $\mathbf{M}$ as the mask, $\mathbf{I}$ as the intensity (aerial) image, $\mathbf{Z}$, $\mathbf{Z}_{\text{in}}$, and $\mathbf{Z}_{\text{out}}$ as the wafer images under nominal process condition $\mathbf{P}_{\text{nom}}$ (nominal dose and nominal focus $\mathbf{H}$), min process condition $\mathbf{P}_{\text{min}}$ (min dose and defocus $\mathbf{H}_{\text{def}}$), and max process condition $\mathbf{P}_{\text{max}}$ (max dose and nominal focus $\mathbf{H}$), respectively. Operators "$\otimes$" and "$\odot$" are used to represent convolution and elementwise product, and $\phi(\cdot;\cdot)$ stands for the forward function of the neural network.

A. Lithography Simulation Model

The lithography simulation models are designed to mimic the printing effects without performing actual lithography. In practice, the Hopkins diffraction model of the partial coherence imaging system [21] is widely adopted to approximate the printing behavior of the lithography (193-nm immersion lithography system with annular illumination in this article). An approximate solution to the Hopkins imaging equations called sum of coherent sources (SOCSs) can be applied to calculate the aerial image $\mathbf{I}$, which is a distribution of light intensity at the wafer plane. Theoretically, the aerial image can be obtained by convolving the mask $\mathbf{M}$ with a set of precomputed coherent convolution kernels $\mathbf{H}$ and taking the weighted

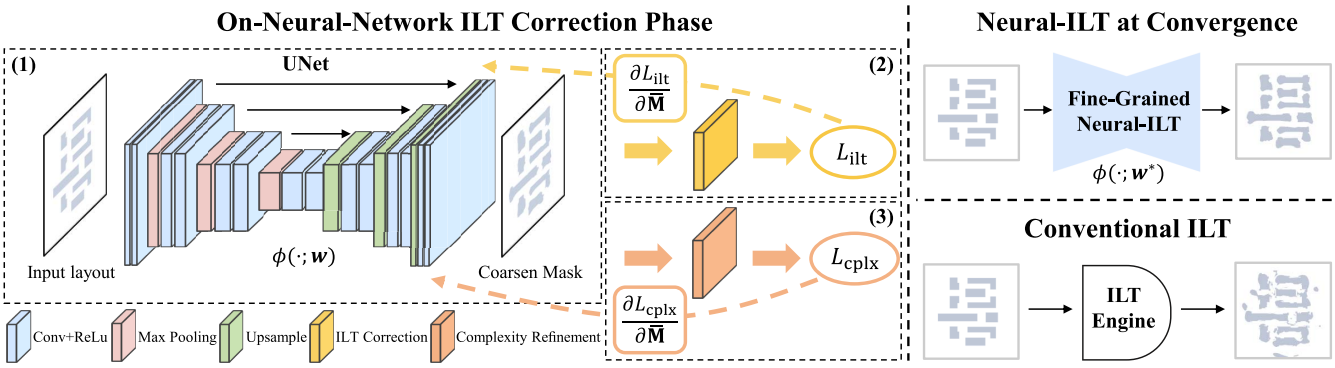


Fig. 2. Overview of Neural-ILT. The on-neural-networks ILT correction is equivalent to the training procedure of Neural-ILT in an *unsupervised* manner. At convergence, the fined-grained Neural-ILT is able to directly generate optimized mask with reasonably good printability and low complexity.

summation with corresponding coefficients $\boldsymbol{\omega}$

$$\mathbf{I}(x,y) = \sum_{k=1}^{N^2} \omega_k |\mathbf{M}(x,y) \otimes \mathbf{h}_k(x,y)|^2 \tag{1}$$

where $\mathbf{h}_k$ is the kth kernel of the model and ω_k is the corresponding weight. The system can be further simplified by an N_hth order approximation [4], where $N_h = 24$ in our implementation. A constant threshold resist (CTR) model that reflects exposure level on the photo resist is then applied to the intensity image to control the final binary wafer image through the following step function:

$$\mathbf{Z}(x,y) = \begin{cases} 1, & \text{if } \mathbf{I}(x,y) \geq I_{\text{th}} \\ 0, & \text{if } \mathbf{I}(x,y) < I_{\text{th}} \end{cases} \tag{2}$$

where $I_{th} = 0.225$ is a constant in our implementation.

B. Mask Manufacturability and Mask Printability

Since ILT naturally generates purely curvilinear features, conventional fracturing method requires a large number of small rectangles to approximate the shape (as shown in Fig. 1), which makes mask writing extremely expensive [7]. Despite the fact that more advanced mask data preparation techniques like multibeam fracturing and model-based mask data preparation (MB-MDP) have been deployed [7], in this article, we will use the *shot count* of *conventional fracturing* (can accurately replicate the shapes) to measure mask complexity and manufacturability. Without loss of generality, reduction of mask complexity should generally benefit various mask fracturing approaches.

Definition 1 (Mask Fracturing Shot Count): Given a mask $\mathbf{M}$, the mask fracturing shot count denotes the number of rectangular shots for accurately replicating the mask shapes.

Mask printability can be measured by squared L_2 error and process variation band (PVBand), which are defined as follows.

Definition 2 (Squared L_2 Error): Given the target layout image $\mathbf{Z}_{\text{t}}$ and the wafer image $\mathbf{Z}$ of a mask $\mathbf{M}$, where $\mathbf{Z} = f(\mathbf{M}; \mathbf{P}_{\text{nom}})$, the squared L_2 error of $\mathbf{Z}$ is given by $||\mathbf{Z} - \mathbf{Z}_{\text{t}}||_2^2$.

Definition 3 (PVBand): PVBand denotes the contour area variations under $\pm 2\%$ dose error, which is measured by the summation of bitwise-XOR between $\mathbf{Z}_{\text{in}}{=}f(\mathbf{M}; \mathbf{P}_{\text{min}})$ and $\mathbf{Z}_{\text{out}}{=}f(\mathbf{M}; \mathbf{P}_{\text{max}})$.

Problem 1 (Learning-Based OPC): Given a target layout $\mathbf{Z}_{\text{t}}$ and a lithography simulation model $f(\cdot; \mathbf{P}_{\text{nom}})$, use learning-based approaches to generate a mask solution $\mathbf{M}_{\text{opt}}$, while simultaneously minimizes: 1) squared L_2 error; 2) PVBand; 3) mask fracturing shot count; and 4) turn around time.

III. Neural-ILT Algorithms

The objective of a typical end-to-end ILT OPC process is to find a mask solution $\mathbf{M}_{\text{opt}} = f^{-1}(\mathbf{Z}_{\text{t}}; \mathbf{P}_{\text{nom}})$ for the given layout $\mathbf{Z}_{\text{t}}$, where $f(\cdot; \mathbf{P}_{\text{nom}})$ is the forward lithography simulation under nominal condition. Since different mask solutions may yield the same wafer image, there are no explicit closed-form formula for solving the inverse lithography process $f^{-1}(\cdot; \mathbf{P}_{\text{nom}})$. Alternatively, the ILT problem can be solved with numerical algorithms like gradient descent, where the gradient derived from the loss function is applied to update the on-mask pixels iteratively until the specific criterion for convergence is met. Regarding the entire ILT process as a black-box, learning a mapping between the input layout and the output mask can be naturally formulated as an image-to-image translation task. More precisely, to mimic the ILT correction process in an end-to-end manner, an encoder–decoder neural network architecture (e.g., UNet) can be applied to perform pixelwise mask prediction.

The overall flow of Neural-ILT is illustrated in Fig. 2. Neural-ILT consists of three modules: 1) a pretrained backbone network; 2) an ILT correction layer; and 3) a mask complexity refinement layer. Given an input layout $\mathbf{Z}_t$ to Neural-ILT, the forward propagation of the backbone network (a pretrained UNet) will first generate a coarsened mask which is then fed into the customized refinement layers. The refinement layers then return corresponding gradients to update the weights of the backbone network through backward propagation. At convergence, Neural-ILT is able to directly generate a well-optimized mask through the forward inference.

In this section, we will detail our proposed framework in a bottom-up manner. A high-performance CUDA-based lithographic simulation tool will be first introduced, followed by the customized ILT functionality layers which serve as the

Algorithm 1 CUDA-Based Lithography Simulation

Require: Mask $\mathbf{M}$, kernels $\mathbf{H}$, weights $\boldsymbol{\omega}$, DOSE and MODE.

function CUDA_LITHO($\mathbf{M}$, $\mathbf{H}$, $\boldsymbol{\omega}$, DOSE, MODE)
  Load kernels $\mathbf{H}$, weights $\boldsymbol{\omega}$ and mask $\mathbf{M}$ into gpu memory;
  $\mathbf{M}_{\text{cplx}} \leftarrow$ Initialize each pixel $\mathbf{M}_{\text{cplx}}(x, y)$ as complex value;
  $\mathbf{M}_{\text{cplx}}.\text{real} \leftarrow \mathbf{M} * \text{DOSE}, \quad \mathbf{M}_{\text{cplx}}.\text{img} \leftarrow \mathbf{0}$;
  $\mathbf{M}_{\text{fft}} \leftarrow \texttt{CUDA\_FFT}(\mathbf{M}_{\text{cplx}})$;
  $\mathbf{I}_{\text{fft}} \leftarrow$ Initialize each pixel $\mathbf{I}_{\text{fft}}(x, y)$ as a complex vector;
  for k=1,..,24 **do**
    Pad $\mathbf{H}_{\text{k}}$ with 0 to the size of $\mathbf{M}_{\text{fft}}$;
    $\mathbf{I}_{\text{k\_fft}}.\text{real} \leftarrow \mathbf{M}_{\text{fft}}.\text{real} \odot \mathbf{H}_{\text{k}}.\text{real} - \mathbf{M}_{\text{fft}}.\text{img} \odot \mathbf{H}_{\text{k}}.\text{img}$;
    $\mathbf{I}_{\text{k\_fft}}.\text{img} \leftarrow \mathbf{M}_{\text{fft}}.\text{img} \odot \mathbf{H}_{\text{k}}.\text{real} + \mathbf{M}_{\text{fft}}.\text{real} \odot \mathbf{H}_{\text{k}}.\text{img}$;
  end for
  $\mathbf{I}_{\text{ifft}} \leftarrow \texttt{CUDA\_IFFT}(\mathbf{I}_{\text{fft}})$;
  if MODE == CONVOLVE **then** ▷ Litho-convolution
    for each pixel $\mathbf{I}_{\text{ifft}}(x, y)$ **do**
      $\mathbf{I}_{sqrt}(x, y) \leftarrow \sum_{k=1}^{24} \boldsymbol{\omega}^{\frac{1}{2}}{}_k * \mathbf{I}_{\text{k\_ifft}}(x, y)$;
    end for
    return Square rooted intensity map $\mathbf{I}_{sqrt}$;
  end if
  if MODE == SIMULATION **then** ▷ Litho-simulation
    for each pixel $\mathbf{I}_{\text{ifft}}(x, y)$ **do**
      $\mathbf{I}(x, y) \leftarrow \sum_{k=1}^{24} \boldsymbol{\omega}_k * \mathbf{I}^2_{\text{k\_ifft}}(x, y)$;
    end for
    $\mathbf{Z} \leftarrow$ Apply resist model in 2 on $\mathbf{I}$;
    return Intensity map $\mathbf{I}$, wafer image $\mathbf{Z}$;
  end if
end function

core engine for achieving on-neural-network ILT correction. Finally, we will build the complete Neural-ILT framework.

A. CUDA-Based Lithography Simulation

Since there is no explicit closed-form solutions for the ILT problem, solving ILT is essentially to minimize the difference between the lithography simulation output and the target layout by modifying the mask. Therefore, from the perspective of neural network learning, being able to integrate the lithography simulation into a neural network framework is the first challenge for building up Neural-ILT. The major challenge comes from the computing overhead of the lithography simulation process. Conventional litho-simulation suffers from severe computational overhead in advanced technology nodes, and multiple rounds of litho-simulation (per clip) are usually indispensable for guiding the mask correction. Thus, it is imperative to derive a fast litho-simulation tool without loss of accuracy.

For the purpose of increasing the algorithmic parallelism, we develop a GPU-based high performance lithography simulation tool with CUDA based on the ICCAD 2013 contest lithography evaluation tool [22]. Algorithm 1 depicts the details of the implementation. The procedure can be described as follows: 1) compute the Fourier coefficients of the mask; 2) compute the convolutions (in frequency domain) and inverse Fourier transform to obtain light intensity on each pixel; and 3) apply the resist model to the intensity image to get the binary wafer image.

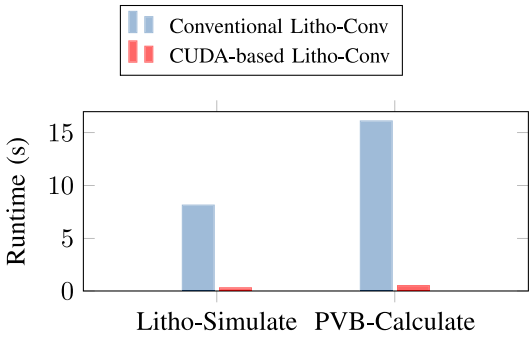


Fig. 3. Runtime comparisons for lithography simulation and PVBand calculation.

As shown in Fig. 3, equipped with the CUDA-based lithography simulation tool, we can achieve more than 96% reduction in litho-simulation time and 97% reduction in PVBand calculation time, which significantly enhances the capability of performing lithography simulation inside neural networks on deep learning platforms like `PyTorch` or `TensorFlow`. In the following parts, we will show how the proposed CUDA-based litho-simulation tool helps to drive the key optimization process of Neural-ILT.

B. Pretrained Backbone Neural Network

Finding a mapping between an input layout and an output mask is usually cast as an image-to-image translation task. In our case, the general encoder–decoder neural network architecture can be applied to make pixelwise binary classification on whether a pixel belongs to the optimized mask. A standard encoder–decoder structure is composed of two subnetworks: 1) an encoder which learns how to compress the input image into an encoded representation and 2) a decoder which learns how to reconstruct an output image from an encoded representation so as to minimize the reconstruction error.

Just like other learning-based solutions [12], [18], inside Neural-ILT, a pretrained backbone model is required to provide the basic layout-to-mask translation functionality. Note that the primary objective of this work is to demonstrate the feasibility and effectiveness of on-neural-network ILT correction, we want to propose a general paradigm to achieve robust and competitive results on most image-to-image translation networks, rather than relying on the use of certain fancy network architectures. Therefore, we selected to use UNet [23], a well-known but relatively simple model, to serve as the basic module of Neural-ILT. As depicted in Fig. 2, the UNet architecture is based on the general encoder–decoder scheme, which consists of a downsampling path to capture context and a symmetric upsampling path that enables pixelwise mask correction. Long skip connections are introduced to bridge the symmetric layers in the downsampling and upsampling paths, which deliver fine-grained details of encoding

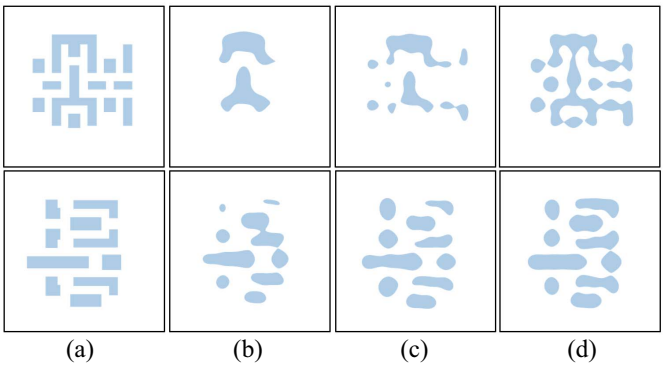


Fig. 4. (a) Target layouts. Wafer images of (b) target layouts, (c) UNet forward predictions, and (d) ILT synthesized masks.

stage that can be recovered in the decoding stage. In general, the superior performance of UNet in image-to-image translation tasks as well as its relatively simple structure make it a suitable choice for conducting the on-neural-network mask correction.

Given a training dataset $\mathcal{D} = \{\mathcal{Z}_t,\ \mathcal{M}^*\}$, where the input target layouts $\mathcal{Z}_t$=$\{\mathbf{Z}_{t,1}, \mathbf{Z}_{t,2}, \ldots, \mathbf{Z}_{t,n}\}$ and the optimized mask labels $\mathcal{M}^*$=$\{\mathbf{M}^*_1, \mathbf{M}^*_2, \ldots, \mathbf{M}^*_n\}$, the training procedure of the UNet is to minimize the following objective:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\ \lambda||\phi(\mathcal{Z}_t; \mathbf{w}) - \mathcal{M}^*||_2^2 \tag{3}$$

where $\phi(\cdot; \mathbf{w})$ is the network forward output w.r.t the model weights $\mathbf{w}$, and λ is a configurable hyperparameter. Fig. 4 shows the image fidelity comparison between the wafer images generated by layouts, pretrained UNet outputs, and ILT synthesized masks. We can see that, the pretrained UNet does improve the mask printability (corresponds to better wafer image fidelity) to some extent compared with the wafer images yielded by the original layouts. However, when dealing with complex layouts, the printability of UNet outputs can hardly be maintained at a satisfactory level, especially in comparison with the ILT synthesized results. In order to compensate for the quality loss introduced by the model prediction error, as well as to maintain the runtime superiority of learning-based solutions, the neural network should be endowed with an ability for self-correction to minimize the lithography error. Such demands indicate the necessity and rationality of our on-neural-network ILT correction paradigm.

C. ILT Correction and Mask Complexity Refinement Layers

Neural-ILT possesses the ability to conduct on-neural-network ILT correction and mask complexity refinement. To enable these features, the functionalities of conventional ILT correction and mask complexity refinement processes are cast as customized neural network layers, which can be directly integrated into an off-the-shelf neural network architecture.

1) ILT Correction Layer: The objective of ILT correction is essentially minimizing the difference between two images

$$L_{\text{ilt}} = \sum_{x=1}^{N}\sum_{y=1}^{N}(\mathbf{Z}(x, y) - \mathbf{Z}_t(x, y))^{\gamma} \tag{4}$$

where $\mathbf{Z}_t$ is the target layout; $\mathbf{Z} = f(\mathbf{M}; \mathbf{P}_{\text{nom}})$ is the wafer image of the mask $\mathbf{M}$; N denotes the image dimension, and γ is a configurable parameter.

In order to obtain the gradient $([\partial L_{\text{ilt}}]/[\partial \mathbf{M}])$ for updating the mask, $\mathbf{Z}$ and $\mathbf{M}$ should be regarded as matrices with continuous values so as to make L_{ilt} differentiable. Here, the binary constraints of $\mathbf{Z}$ and $\mathbf{M}$ are commonly relaxed by the sigmoid function so that the variables become unconstrained. To achieve this, we first introduce an intermediate variable $\bar{\mathbf{M}}$ to bridge the backbone network output and ILT correction layer (via the chain rule). The original binary mask $\mathbf{M}$ and wafer image $\mathbf{Z}$ are then replaced with their sigmoid approximations $\mathbf{M} = \text{sig}(\bar{\mathbf{M}}) = ((1)/[1 + \exp(-\theta_M \times \bar{\mathbf{M}})])$ and $\mathbf{Z} = \text{sig}(\mathbf{I}) = ((1)/[1 + \exp(-\theta_Z \times (\mathbf{I} - I_{th}))])$, where θ_M and θ_Z define the steepness of the sigmoid functions used for $\mathbf{M}$ and $\mathbf{Z}$, respectively. As a result, the gradient of the lithography loss (4) is then given by

$$\begin{aligned}\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}} &= \gamma \times (\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \frac{\partial \mathbf{Z}}{\partial \bar{\mathbf{M}}} \\ &= \gamma \times (\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \frac{\partial \mathbf{Z}}{\partial \mathbf{I}} \odot \frac{\partial \mathbf{I}}{\partial \mathbf{M}} \odot \frac{\partial \mathbf{M}}{\partial \bar{\mathbf{M}}}\end{aligned} \tag{5}$$

where $(\partial \mathbf{I})/(\partial \mathbf{M})$ in (5) can be derived as

$$\begin{aligned}\frac{\partial \mathbf{I}(i,j)}{\partial \mathbf{M}(x,y)} &= \frac{\partial\{|\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)|^2\}}{\partial \mathbf{M}(x,y)} \\ &= \frac{\partial\{[\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)][\mathbf{M}(i,j) \otimes \mathbf{H}^*(i,j)]\}}{\partial \mathbf{M}(x,y)} \\ &= \frac{\partial[\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)]}{\partial \mathbf{M}(x,y)}[\mathbf{M}(i,j) \otimes \mathbf{H}^*(i,j)] \\ &\quad + \frac{\partial[\mathbf{M}(i,j) \otimes \mathbf{H}^*(i,j)]}{\partial \mathbf{M}(x,y)}[\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)] \\ &= \mathbf{H}(i-x, k-y)[\mathbf{M}(i,j) \otimes \mathbf{H}^*(i,j)] \\ &\quad + \mathbf{H}^*(i-x, k-y)[\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)]\end{aligned} \tag{6}$$

here $|\mathbf{M}(i,j) \otimes \mathbf{H}(i,j)|^2 = \sum_{k=1}^{N_h} \omega_k |\mathbf{M}(i,j) \otimes \mathbf{h}_k(i,j)|^2$, and $\mathbf{H}^*$ is the conjugate transpose of kernel $\mathbf{H}$. Substituting (6) into (5), we have

$$\begin{aligned}\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}} &= \{\mathbf{H}^{\text{flip}} \otimes [(\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z}) \odot (\mathbf{M} \otimes \mathbf{H}^*)] \\ &\quad + (\mathbf{H}^{\text{flip}})^* \otimes [(\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z}) \odot (\mathbf{M} \otimes \mathbf{H})]\} \\ &\quad \odot \mathbf{M} \odot (1 - \mathbf{M}) \times \gamma \theta_M \theta_Z\end{aligned} \tag{7}$$

where $\mathbf{H}^{\text{flip}}$ is the 180° rotation of $\mathbf{H}$.

Considering that the loss and gradient computations of ILT correction share similar mechanisms of neural network forward and backward propagation, conventional ILT functionality can be implemented as a customized ILT correction layer of neural network. Regarding L_{ilt} as the forward propagation loss, and $(\partial L_{\text{ilt}})/(\partial \bar{\mathbf{M}})$ as the backward propagation gradient, the weights of the predecessor neural networks $\mathbf{w}_{\text{net}}$ can be updated through the chain rule $[(\partial L_{\text{ilt}})/(\partial \mathbf{M})]$ $[(\partial \mathbf{M})/(\partial \bar{\mathbf{M}})]$ $[(\partial \bar{\mathbf{M}})/(\partial \phi(\mathbf{Z}_t; \mathbf{w}_{\text{net}}))]$ $[(\partial \phi(\mathbf{Z}_t; \mathbf{w}_{\text{net}}))/(\partial \mathbf{w}_{\text{net}})]$. Moreover, thanks to the `CUDA` implementation of the litho-simulation function (Algorithm 1), the entire forward and backward procedures can be perfectly integrated into the unified `CUDA`-compatible deep learning toolkit like `PyTorch` to fully leverage its computational efficiency. Algorithm 2 depicts how

Algorithm 2 ILT Correction Layer Forward and Backward

Require: Masks $\mathbf{M}$, $\bar{\mathbf{M}}$, layout $\mathbf{Z}_t$, kernels $\mathbf{H}$, $\mathbf{H}^*$, weights $\boldsymbol{\omega}$.

function FORWARD($\mathbf{M}$, $\mathbf{H}$, $\boldsymbol{\omega}$)
  $\mathbf{I}, \mathbf{Z} \leftarrow$ CUDA_LITHO($\mathbf{M}$, $\mathbf{H}$, $\boldsymbol{\omega}$, 1.0, SIMULATION);
  $L_{\text{ilt}} \leftarrow ||\mathbf{Z} - \mathbf{Z}_t||_\gamma^\gamma$; ▷ $\gamma = 4$ in forward
  return Lithography loss L_{ilt};
end function
function BACKWARD($\mathbf{M}$, $\bar{\mathbf{M}}$, $\mathbf{H}$, $\mathbf{H}^*$, $\boldsymbol{\omega}$) ▷ $\theta_M = 4$, $\theta_Z = 50$
  $\mathbf{I}, \mathbf{Z} \leftarrow$ CUDA_LITHO($\mathbf{M}$, $\mathbf{H}$, $\boldsymbol{\omega}$, 1.0, SIMULATION);
  $\mathbf{Z} \leftarrow \frac{1}{1+\exp(-\theta_Z \times (\mathbf{I} - I_{th}))}$, $\mathbf{M} \leftarrow \frac{1}{1+\exp(-\theta_M \times \bar{\mathbf{M}})}$;
  Define common term as $\mathbf{T}_C$, gradient left term as $\mathbf{G}_L$, gradient right term as $\mathbf{G}_R$;
  $\mathbf{T}_C \leftarrow (\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z})$;
  $\mathbf{G}_L \leftarrow \mathbf{T}_C \odot$ CUDA_LITHO($\mathbf{M}$, $\mathbf{H}^*$, $\boldsymbol{\omega}$, 1.0, CONVOLVE);
  $\mathbf{G}_R \leftarrow \mathbf{T}_C \odot$ CUDA_LITHO($\mathbf{M}$, $\mathbf{H}$, $\boldsymbol{\omega}$, 1.0, CONVOLVE);
  $\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}} \leftarrow \gamma\theta_M\theta_Z \times$ [CUDA_LITHO($\mathbf{G}_L$, $\mathbf{H}^{\text{flip}}$, $\boldsymbol{\omega}$, 1.0, CONVOLVE) + CUDA_LITHO($\mathbf{G}_R$, $(\mathbf{H}^{\text{flip}})^*$, $\boldsymbol{\omega}$, 1.0, CONVOLVE)] $\odot \mathbf{M} \odot (1 - \mathbf{M})$; ▷ Compute Equation (7) using Algorithm 1
  return Gradient $\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}}$;
end function

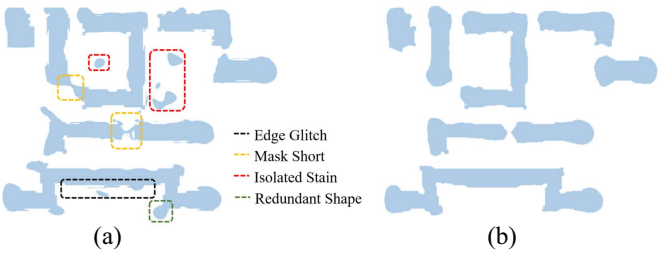


Fig. 5. Mask complexity comparison: (a) ILT synthesized mask and (b) mask purified by complexity refinement.

to implement the forward and backward computations of the ILT correction layer leveraging Algorithm 1.

2) Mask Complexity Refinement Layer: Masks synthesized by ILT usually consist of nonrectangular complex shapes which are not manufacturing-friendly. For conventional ILT which directly updates the mask with gradient descent method, the initial solution can affect the final synthesized mask significantly. As illustrated in Fig. 5(a), the ILT process may generate complex features like isolated curvilinear stains, edge glitches, and redundant contours grown along the existing mask shapes. Therefore, in this work, the optimization of mask complexity is defined as eliminating the nonmanufacturing-friendly complex features while maintaining competitive mask printability.

It is observed that most of these complex features are distributed around/on the original layout patterns [Fig. 5(a)]. They usually will not be printed on wafer image under the *min process* condition $\mathbf{P}_{\min}$ and the *nominal process* condition $\mathbf{P}_{\text{nom}}$, but are likely to be printed at the *max process* condition $\mathbf{P}_{\max}$. This causes the area variations between $\mathbf{Z}_{\text{in}}=f(\mathbf{M}; \mathbf{P}_{\min})$ and $\mathbf{Z}_{\text{out}}=f(\mathbf{M}; \mathbf{P}_{\max})$ with respect to the locations of these features. Therefore, we can formulate the mask complexity refinement task into minimizing the following loss function L_{cplx}:

$$L_{\text{cplx}} = ||\mathbf{Z}_{\text{in}} - \mathbf{Z}_{\text{out}}||_2^2. \tag{8}$$

Similarly, the gradient of loss L_{cplx} can be derived as

$$\frac{\partial L_{\text{cplx}}}{\partial \bar{\mathbf{M}}} = 2 \times (\mathbf{Z}_{\text{in}} - \mathbf{Z}_{\text{out}}) \odot (\mathbf{Z}_{\text{in}'} - \mathbf{Z}_{\text{out}'}). \tag{9}$$

Here, $\mathbf{Z}_{\text{in}'}$ is given by

$$\begin{aligned}\mathbf{Z}_{\text{in}'} = \theta_M\theta_Z \times \{&\mathbf{H}_{\text{def}}^{\text{flip}} \otimes \left[\mathbf{Z}_{\text{in}} \odot (1 - \mathbf{Z}_{\text{in}}) \odot \left(\mathbf{M} \otimes \mathbf{H}_{\text{def}}^*\right)\right] \\ &+ \left(\mathbf{H}_{\text{def}}^{\text{flip}}\right)^* \otimes [\mathbf{Z}_{\text{in}} \odot (1 - \mathbf{Z}_{\text{in}}) \odot (\mathbf{M} \otimes \mathbf{H}_{\text{def}})]\} \\ &\odot \mathbf{M} \odot (1 - \mathbf{M})\end{aligned}$$

where $\mathbf{H}_{\text{def}}$ is the defocus kernels, and $\mathbf{Z}_{\text{out}'}$ can be obtained by a similar derivation. Just like the ILT correction layer, we can cast the above optimization process as a customized mask complexity refinement layer. The detailed implementation of the layer forward/backward computation is omitted here as it is similar to Algorithm 2. The effectiveness of the mask complexity refinement layer is demonstrated in Fig. 5, where the mask in Fig. 5(b) enjoys lower mask complexity and competitive mask printability comparing to the one in Fig. 5(a). Note that, from the perspective of deployment, we should always connect the complexity refinement layer with an ILT correction layer to form a multiobjective loss function (10). Otherwise, optimization with only mask complexity refinement layer will eliminate all mask shapes which literally yield "0" loss for (8).

D. Neural-ILT for Mask Printability and Complexity Co-Optimization

As presented in Fig. 2, the Neural-ILT network model is composed of three modules: 1) a pretrained backbone UNet for performing layout-to-mask transformation; 2) an ILT correction layer for minimizing lithography loss; and 3) a mask complexity refinement layer for removing redundant complex features. Consequently, the on-neural-network ILT correction is equivalent to an unsupervised training procedure of Neural-ILT with the following objective:

$$\begin{aligned}\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}} \; &\alpha \overbrace{||f(\phi(\mathbf{Z}_t; \mathbf{w}); \mathbf{P}_{\text{nom}}) - \mathbf{Z}_t||_\gamma^\gamma}^{L_{\text{ilt}}} \\ &+ \beta \underbrace{||f(\phi(\mathbf{Z}_t; \mathbf{w}); \mathbf{P}_{\min}) - f(\phi(\mathbf{Z}_t; \mathbf{w}); \mathbf{P}_{\max})||_2^2}_{L_{\text{cplx}}}\end{aligned} \tag{10}$$

where α, β, and γ are the configurable hyperparameters.

The backbone network $\phi(\cdot; \mathbf{w})$ of the Neural-ILT can be replaced by any other network architectures that can yield image output with the required dimensions (2048 × 2048 in Neural-ILT). Conceptually, we can update the network weights $\mathbf{w}$ by backward propagation through the chain rule

$$\frac{\partial L_{\text{refine}}}{\partial \mathbf{w}} = \frac{\partial L_{\text{refine}}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \bar{\mathbf{M}}} \frac{\partial \bar{\mathbf{M}}}{\partial \phi(\mathbf{Z}_t; \mathbf{w})} \frac{\partial \phi(\mathbf{Z}_t; \mathbf{w})}{\partial \mathbf{w}} \tag{11}$$

where $L_{\text{refine}} = \alpha \cdot L_{\text{ilt}} + \beta \cdot L_{\text{cplx}}$ denotes the refine loss and $\mathbf{Z}_t$ denotes the layout input.

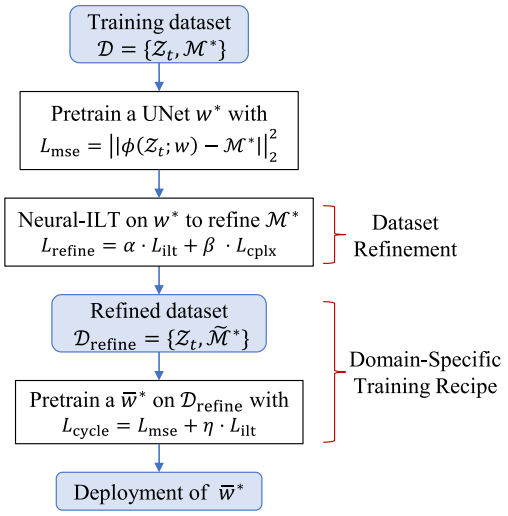


Fig. 6. Diagram of Neural-ILT training recipe, from pretraining to deployment.

E. Pretrain Backbone Network With Domain Knowledge of Partial Coherent Imaging System

In order to improve the training quality of our Neural-ILT, we propose a domain-specific training recipe, which comprises of a dataset refinement stage and domain-specific pretraining stage. Fig. 6 depicts the details of our training recipe.

The original training dataset $\mathcal{D}\{\mathcal{Z}_t,\ \mathcal{M}\}$ is composed of 4300 pairs of target layout and reference mask, and the reference masks are synthesized by a conventional ILT tool [4]. However, as shown in Fig. 5(a), ILT synthesized masks usually consist of numerous complex features, which may results in an initial prediction with high mask complexity for the Neural-ILT. As shown in Fig. 6, in order to improve the training data quality, we will first pretrain a UNet $\mathbf{w}^*$ on the dataset $\mathcal{D}$ using (3) in the dataset refinement stage. Neural-ILT (10) is then conducted on $\mathbf{w}^*$ to refine the complex features in $\mathcal{M}$. The resulted masks $\tilde{\mathcal{M}}^*$ [e.g., Fig. 5(b)] together with the original target layouts $\mathcal{Z}_t$ will form a novel refined dataset $\mathcal{D}_{\text{refine}} = \{\mathcal{Z}_t,\ \tilde{\mathcal{M}}^*\}$.

In the domain-specific pretraining stage, we can use this refined dataset $\mathcal{D}_{\text{refine}}$ to pretrain a new UNet with the following cycle loss L_{cycle}:

$$L_{\text{cycle}} = \overbrace{||\phi(\mathcal{Z}_t; \mathbf{w}) - \tilde{\mathcal{M}}^*||_2^2}^{\text{Supervised term}} + \eta \overbrace{||f(\phi(\mathcal{Z}_t; \mathbf{w}); \mathbf{P}_{\text{nom}}) - \mathcal{Z}_t||_2^2}^{\text{Domain-specific regularization (DSR) term}} \quad (12)$$

where $\mathcal{Z}_t$ and $\tilde{\mathcal{M}}^*$ refer to the input target layouts and ILT mask labels in the refined dataset, respectively; η is configurable. The calculations of the first and second terms in L_{cycle} form a closed loop. The first supervised term minimizes the image difference between network predictions and labels. The second term is essentially the ILT loss in (10), in which the *domain knowledge* of the partial coherent imaging model is introduced into the network training procedure by the litho-simulation function $f(\cdot; \mathbf{P}_{\text{nom}})$. As a result, the training process simultaneously considers both the supervised loss and ILT objective. The second term essentially serves as a domain-specific regularization (DSR) term, which guides a newly pretrained network $\phi(\cdot; \mathbf{w})$ to gradually converge toward a more ILT-compatible direction.

F. Multitask-Enabled Neural-ILT

Neural-ILT is highly modular which offers flexibility in: 1) modifying the network architecture and training strategy; 2) reconfiguring the lithographic recipes; 3) adding customized objectives during ILT optimization, etc. Through the recombination of existing capabilities and modified modules, Neural-ILT is capable of achieving rapid adaptation to conduct mask correction on new designs that developed in other technology nodes, thereby reducing the development overhead.

To verify the feasibility of the above claim, we further develop Neural-ILT-multitask, a proof-of-concept (POC) prototype based on the original Neural-ILT. As shown in Fig. 7, Neural-ILT-multitask aims to provide end-to-end mask optimizations for different datasets (i.e., designs with different styles and technology nodes) on a common pretrained backbone network $\mathbf{w}$. In this article, besides the regular ILT correction for 32-nm metal-layer layouts, Neural-ILT-multitask is also capable of minimizing the edge placement error (EPE) violations for 45-nm cut-layer layouts.

There are three factors that distinguish the cut-layer task from the conventional metal-layer one. First, the 45-nm cut-layer's lithography recipe is different from the 32-nm metal-layer's recipe. Second, in cut-layer mask optimization, the minimization of the EPE (see Definition 4) is the most critical objective to ensure the printability of the via patterns. Third, as visualized in Fig. 7(b), the characteristics of cut-layer's and metal-layer's layouts varies significantly, which indicates the difficulty of handling both datasets within a same backbone network. As a result, modifications on: 1) backbone network training strategy; 2) lithography model configurations; and 3) layer functionality are indispensable to carry out the cut-layer mask optimization.

Definition 4 (EPE): EPE measures the horizontal and vertical edge displacement between the printed contour and the target layout by a set of on-edge checkpoints (see in Fig. 8). A checkpoint will be marked as an EPE violation (EPEV) if its EPE exceeds a given threshold th_{epe}.

1) Fast EPE Correction Heuristic: Instead of counting the cumulative error among all image pixels like the ILT loss in (4), the minimization of EPE only focuses on the error within particular checking segments. As illustrated in Fig. 8, a set of horizontal/vertical EPE checkpoints (HC/VC) are evenly distributed on the horizontal/vertical edges of the target patterns. A horizontal/vertical checking segment (HS/VS) expands the vertical/horizontal checkpoint (VC/HC) by $\pm th_{\text{epe}}$ pixels

$$\begin{aligned} \text{HS} &= [(i - \text{th}_{\text{epe}}, \text{j}), (\text{i} + \text{th}_{\text{epe}}, \text{j})], \ \text{if } (\text{i}, \text{j}) \in \text{VC} \\ \text{VS} &= [(i, j - \text{th}_{\text{epe}}), (\text{i}, \text{j} + \text{th}_{\text{epe}})], \ \text{if } (\text{i}, \text{j}) \in \text{HC}. \end{aligned}$$

An EPEV will be introduced if the cumulative error (pixel differences) within a specific checking segment exceeds the

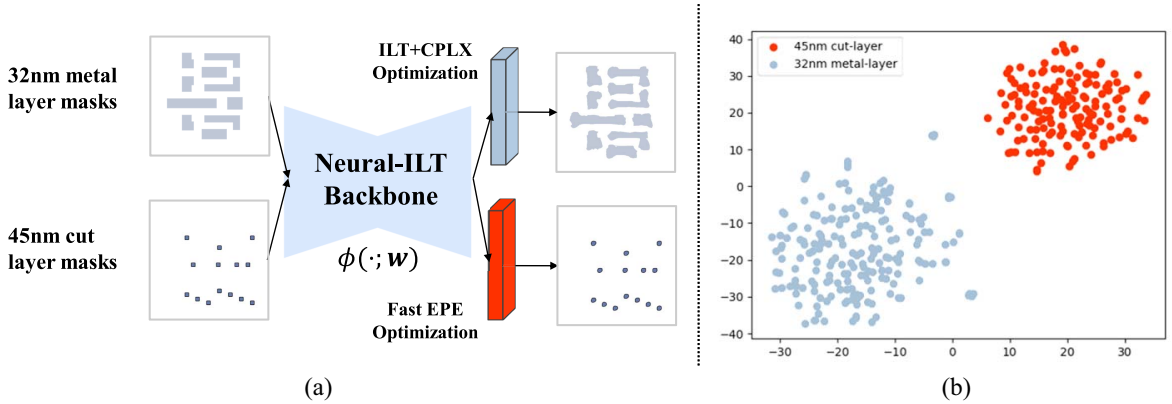


Fig. 7. (a) Overview of multitask-enabled Neural-ILT framework. (b) T-distributed stochastic neighbor embedding (T-SNE) visualization of two datasets, each dataset forms a unique cluster, which indicates each dataset possesses its own characteristic.

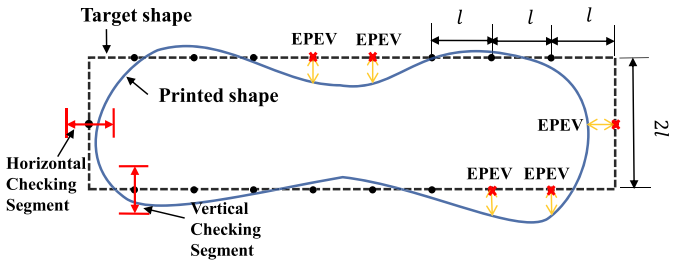


Fig. 8. Illustration of EPE measurement. The number of EPEV for this printed contour is 5.

predefined threshold th_{epe}, that is

$$\mathrm{D}_{\text{epe}(i,j)} = \sum_{i-\text{th}_{\text{epe}}}^{i+th_{\text{epe}}} (\mathbf{Z}(i,j) - \mathbf{Z}_{\text{t}}(i,j))^2, \text{ if } (i,j) \in \text{VC}$$

$$\mathrm{D}_{\text{epe}(i,j)} = \sum_{j-\text{th}_{\text{epe}}}^{j+th_{\text{epe}}} (\mathbf{Z}(i,j) - \mathbf{Z}_{\text{t}}(i,j))^2, \text{ if } (i,j) \in \text{HC}$$

$$\text{EPEV}_{(i,j)} = \begin{cases} 1, & \text{if } \mathrm{D}_{\text{epe}(i,j)} \geq th_{\text{epe}} \\ 0, & \text{if } \mathrm{D}_{\text{epe}(i,j)} < th_{\text{epe}}. \end{cases} \tag{13a}$$

We further develop a simple yet effective heuristic to simultaneously achieve fast and efficient EPE minimization. Our fast EPE correction leverages the existing ILT correction layer utility, where the losses on the region of interest for EPE checking are specially enhanced by a scale factor δ, that is

$$L_{\text{epe\_fast}} = \left(1 + \mathbf{R}_{\text{epe}}\right) \odot (L_{\text{ilt}} + P_{\text{discrete}}) \tag{14}$$

where P_{discrete} is a mask discrete penalty term introduced in [26], $\mathbf{R}_{\text{epe}}$ is the region of interest for EPE checking

$$\mathbf{R}_{\text{epe}}(i,j) = \begin{cases} \delta, & \text{if } (i,j) \in \{HS, VS\} \\ 0, & \text{else} \end{cases} \tag{15}$$

and δ is configurable.

The effectiveness of our fast EPE correction heuristic can be verified in Fig. 9 on FreePDK45 [25] benchmarks, where the average number of EPEVs dropped significantly to a comparable SOTA level [24] (even better) as the value of δ increased, while the L_2 loss remains stable, which is much lower than the baseline. Moreover, unlike $\nabla L_{\text{epe\_exact}}$ calculation, there is

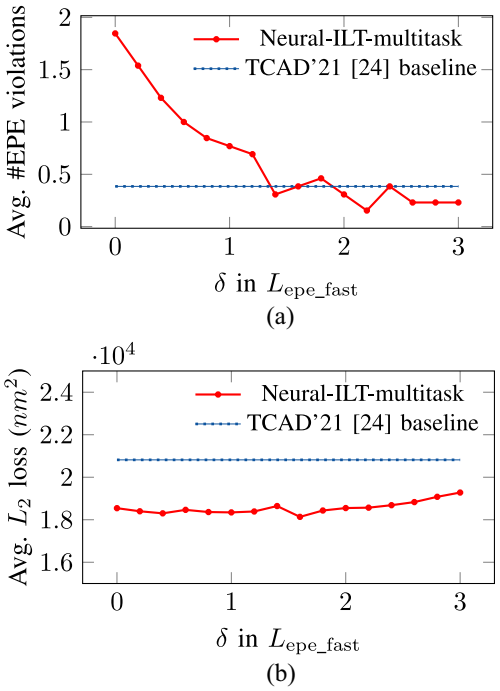


Fig. 9. Selection of parameter δ in $L_{\text{epe\_fast}}$ (14). The average #EPEVs of Neural-ILT-multitask on FreePDK45 [25] benchmarks get significantly reduced as δ increased, while not changing much L_2 loss. (a) Average #EPEVs w.r.t. different δ. (b) Average L_2 loss w.r.t. different δ.

nearly no additional overhead for calculating $\nabla L_{\text{epe\_fast}}$ beyond the ∇L_{ilt} calculation.

2) Backbone Model Pretraining for Neural-ILT-Multitask: In this article, Neural-ILT-multitask aims to simultaneously conduct ILT correction for 32-nm metal-layer masks [through the blue layer in Fig. 7(a)] and EPE correction for 45-nm cut-layer masks [through the red layer in Fig. 7(a)] within the same pretrained backbone model $\mathbf{w}$. However, the characteristics of the above two datasets vary significantly [see in Fig. 7(b)]. Therefore, our backbone model $\mathbf{w}$ should be able to generate proper responses for inputs from different tasks. To this end, we first pretrain a backbone model $\mathbf{w}$ (6 epochs) to minimize the cycle loss in (12) on the 32-nm metal-layer dataset $\{\mathcal{Z}_t, \tilde{\mathcal{M}}^*\}$. Then, we apply an additional round of fine-tuning

TABLE I
PERFORMANCE COMPARISONS WITH SOTA METHODS

Cut-layer Mask Optimization Task on FreePDK45							Metal-layer Mask Optimization Task on ICCAD'13 Benchmarks													
Case	TCAD'21 [24]			Ours (Multitask)			Case	Model-based [1]			ILT-based [4]			PGAN-OPC [18]			Ours		
	L_2	PVB	TAT	L_2	PVB	TAT		L_2	PVB	TAT	L_2	PVB	TAT	L_2	PVB	TAT	L_2	PVB	TAT
AND2_X1	18032	-	282	16645	35581	34	1	53816	66218	278	51418	58232	318	52570	56267	358	49817	55975	11
AOI211_X1	26439	-	292	21515	46737	34	2	41382	53434	142	40003	47139	256	42253	50822	368	38174	52010	17
BUF_X1	14927	-	170	13993	28062	32	3	79255	146776	152	97589	82195	321	83663	94498	368	89411	91357	10
CLKBUF_X1	12267	-	200	12662	27068	32	4	21717	33266	307	23307	28244	322	19965	28957	377	16744	29982	9
INV_X1	10427	-	181	9217	22980	31	5	48858	65631	189	47755	56253	315	44733	59328	369	45598	58900	11
NAND3_X2	26522	-	194	29751	54977	32	6	46320	62068	353	44471	50981	314	46062	52845	364	43836	54969	10
NAND4_X1	22409	-	178	18592	42640	33	7	31898	51069	219	32713	46309	239	26438	47981	377	20324	50542	16
NOR2_X1	18488	-	278	13022	28941	31	8	23312	25898	99	19987	22482	258	17690	23564	383	13337	26353	15
NOR3_X2	25745	-	357	21717	47831	32	9	55684	75387	119	56252	65331	322	56125	65417	383	49401	68817	11
OAI211_X1	23853	-	196	21188	46819	19	10	19722	18536	61	14518	18868	231	9990	19893	366	8511	20734	14
OAI33_X1	29081	-	294	21114	51979	32													
OR2_X1	18409	-	281	15804	34171	32													
OR4_X1	24026	-	205	20127	43458	32													
Avg.	20817.3	-	239.1	**18103.6**	39326.5	**31.3**	Avg.	42196.4	59828.3	191.9	42801.3	**47603.4**	289.6	39948.9	49957.2	371.3	**37515.3**	50963.9	**12.4**
Ratio	1.00	-	1.00	**0.87**	-	**0.13**	Ratio	1.00	1.00	1.00	1.01	**0.80**	1.51	0.95	0.84	1.93	**0.89**	0.85	**0.06**

Unit: L_2 (nm^2), PVB (nm^2), TAT (s). PVB of TCAD'21 [24] is not available.

(6 epochs) on the same backbone model to minimize the following EPE-guided cycle loss on the 45-nm cut-layer dataset $\{\mathcal{Z}_t^{45\text{nm}},\ \mathcal{M}_{45\text{nm}}^*\}$

$$L_{\text{cycle\_epe}} = \overbrace{||\phi(\mathcal{Z}_t^{45\text{nm}};\mathbf{w}) - \mathcal{M}_{45\text{nm}}^*||_2^2}^{\text{Supervised term}} + \overbrace{L_{\text{epe\_fast}}}^{\text{DSR term}} \quad (16)$$

where $L_{\text{epe\_fast}}$ still serves as the DSR term to introduce the domain knowledge of partial coherent lithography system. In our implementation, δ is set to 2 for both training and testing.

IV. EXPERIMENTAL RESULTS

The proposed Neural-ILT framework is developed with `PyTorch` and `CUDA`, and tested on a Linux machine with 2.2-GHz Intel Xeon CPU and a single Nvidia RTX3090 GPU. The training dataset for 32-nm metal-layer is obtained from the authors of GAN-OPC [18], which contains 4300 training instances based on the design specifications from the existing 32-nm M1 layout topologies. While the training dataset for 45-nm cut-layer is obtained from Zhong *et al.* [24] which contains 150 training instances based on the design specifications from 45-nm topologies in [25]. For Neural-ILT, the UNet is pretrained for 20 epochs which takes around 8 h on a single RTX3090 GPU. As for Neural-ILT-multitask, the UNet is pretrained for six epochs in each task, which takes around 3 h on a single RTX3090 GPU.

We test the Neural-ILT ("Ours") on ICCAD 2013 contest IBM benchmarks [22] to verify their effectiveness on 32-nm metal-layer ILT correction task. In addition, we test Neural-ILT-multitask ["Ours (Multitask)"] on the FreePDK45 [25] benchmarks to verify its effectiveness on 45-nm cut-layer EPE Minimization task. As for the evaluation, the lithography recipes and quality checker are provided by: 1) the ICCAD 2013 contest evaluation package [22] for 32-nm M1 layout designs and 2) Zhong *et al.* [24] for FreePDK45 45-nm cut-layer designs.[1] The mask fracturing tool is implemented with `C++` programming language based on an efficient contour decomposition algorithm in [27].

A. Evaluation of Neural-ILT

To verify the effectiveness of our proposed framework (denoted as "Ours"), we optimize ten industrial 32-nm M1 layout masks (blind for model training, size of mask is 2048×2048) in ICCAD 2013 contest benchmark suite [22] and compare the results with the model-based OPC [1] (denoted as "model-based"), the conventional ILT method [4] (denoted as "ILT"), and the learning-based GAN-OPC method [18] (denoted as "PGAN-OPC"). Quantitative results are listed in Table I, where column "TAT" lists the turnaround time for the entire end-to-end mask optimization on the given input mask; columns "L_2," and "PVB" denote the squared L_2 error and the PVBand, respectively.

1) Hyper-Parameters and Update Criterion Selection: During the on-neural-network ILT correction stage, we use a consistent configuration for every test input. We select Adam as the optimizer, where the maximum ILT iterations = 60, initial learning rate lr = 0.002, lr decay rate = 0.1, lr decay step size = 35 iterations, $\alpha = 1$, and $\beta = 1.45$ [for (10)]. Early stop will be triggered if the objective does not improve for consecutive 15 iterations. We studied the impact of hyperparameter β in the Neural-ILT objective $L_{\text{refine}} = \alpha \cdot L_{\text{ilt}} + \beta \cdot L_{\text{cplx}}$. Consider that sweeping both α and β introduces too large searching space, we fixed $\alpha = 1.0$ and only swept the β from 0 to 4.0 with a stepsize of 0.05 on the ICCAD'13 benchmarks. The corresponding performance tradeoff profiles are presented in Fig. 10.

Moreover, three types of criteria for updating the best solution are compared in Fig. 10.

[1]To the best of our knowledge, above two benchmark suites are the only two open-sourced packages with lithography simulation recipes provided.

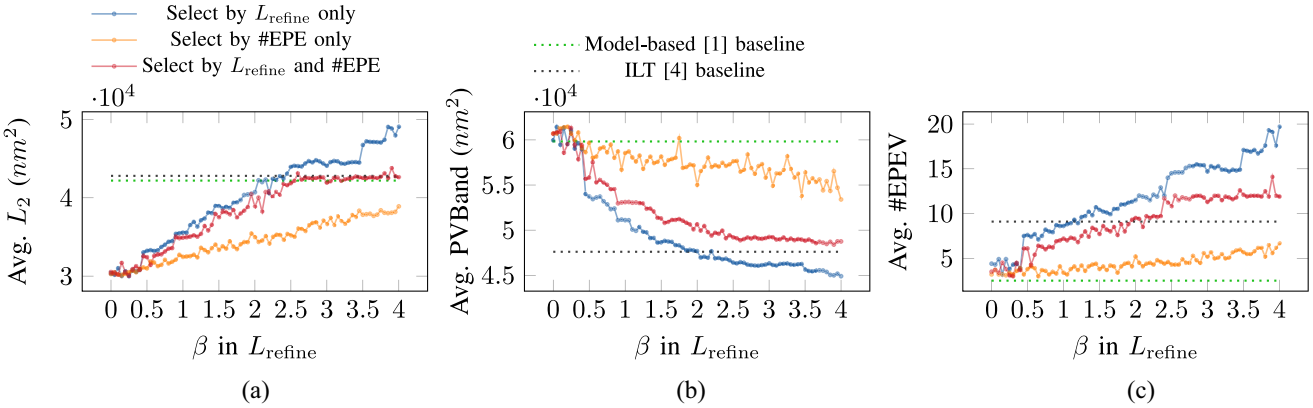


Fig. 10. Printability profiles (L_2 loss, PVBand, #EPEV) of Neural-ILT w.r.t. different values of β in L_{refine} (10).

TABLE II
EPE AND MASK COMPLEXITY COMPARISONS WITH SOTA METHODS

Cut-layer Mask Optimization Task on FreePDK45					Metal-layer Mask Optimization Task on ICCAD'13 Benchmarks								
Case	TCAD'21 [24]		Ours (Multitask)		Case	Model-based [1]		ILT-based [4]		PGAN-OPC [18]		Ours	
	#EPE	#Shots	#EPE	#Shots		#EPE	#Shots	#EPE	#Shots	#EPE	#Shots	#EPE	#Shots
AND2_X1	1	-	0	182	1	0	242	6	1382	8	931	8	428
AOI211_X1	0	-	2	219	2	0	208	10	755	13	692	3	256
BUF_X1	0	-	0	126	3	18	320	59	1279	51	1048	52	557
CLKBUF_X1	0	-	0	141	4	0	93	1	591	2	386	2	136
INV_X1	0	-	0	113	5	1	293	6	980	8	950	3	380
NAND3_X2	0	-	1	320	6	0	319	1	1379	12	836	5	383
NAND4_X1	0	-	0	199	7	0	185	0	788	7	515	0	244
NOR2_X1	0	-	0	143	8	0	139	2	497	0	286	0	285
NOR3_X2	1	-	1	215	9	1	327	6	1575	12	1087	2	444
OAI211_X1	1	-	0	222	10	0	81	0	531	0	338	0	208
OAI33_X1	1	-	0	258									
OR2_X1	1	-	0	179									
OR4_X1	0	-	0	222									
Avg.	0.4	-	**0.3**	195.3	Avg.	**2.0**	**220.7**	9.1	975.7	11.3	706.9	7.5	332.1
Ratio	1.00	-	**0.80**	-	Ratio	**1.00**	**1.00**	4.55	4.31	5.65	3.20	3.75	1.50

Unit: L_2 (nm^2), PVB (nm^2), TAT (s). #Shots of TCAD'21 [24] is not available.

1) *Select by L_{refine} Only:* Let $score = \alpha \cdot L_{\text{ilt}} + \beta \cdot L_{\text{cplx}}$, update the best solution if *currernt_score* < *best_score*.
2) *Select by #EPEV Only:* Let #EPEV stands for number of EPEVs, update the best solution if #*currernt_epev* < #*best_epev*.
3) *Select by L_{refine} and #EPEV:* Update the best solution if (*currernt_score* < *best_score*) and (#*currernt_epev* ≤ #*best_epev*).

It is obvious that different criteria generated significantly different tradeoff curves, especially on the EPE metric. An interesting observation here is that, in the coarse-grained level, EPE curve shares the same trend as the L_2 curve. But in the fine-grained level, or in a local range, a better L_2 may not necessarily reflect a better EPE, and vice versa.

For Neural-ILT, we adopt the criterion of "select by L_{refine} and #EPEV" (red lines) to consider L_2, PVB, and EPE simultaneously, and we set $\beta = 1.45$ according to the profiles.

2) Performance Comparisons With SOTA Methods: In Table I, it can be observed that our framework outperforms all other SOTA approaches in terms of "TAT" and "L_2" metrics. More specifically, comparing with model-based OPC [1], ILT [4] and PGAN-OPC [18], we achieve more than 15×, 23×, 30× turnaround time (TAT) speedups, with 12.4%, 14.1%, and 6.5% L_2 error reductions and comparable PVBands.

As for the EPE performance, it can be observed from Table II, our EPE result outperforms the ILT and GAN-OPC baselines with 1.6 and 3.8 less average EPEVs, while the model-based method demonstrating the best EPE performance. However, model-based method consumes 15% more PVBand to achieve this EPE and yields the worst L_2+PVBand score among all methods (see in Fig. 11). As mentioned earlier, EPE approximates the L_2 loss with a discrete measurement, but they may not be able to precisely reflect each other. As a result, a better EPE result may not necessarily represent a better printability in terms of L_2 and PVBand. According to the printability profiles in Fig. 10(a)–(c), our tool can also achieve comparable average EPE and PVBand results as the model-based method by setting β within the range of [0, 0.5], while enjoying a much lower average L_2 error.

3) Mask Complexity Comparisons: Table II compares the complexity of masks (mask shots count) synthesized by the model-based method, ILT method, GAN-OPC family, and our Neural-ILT, and Fig. 12 visualizes their synthesized masks on four testcases. As expected, Neural-ILT family is able to

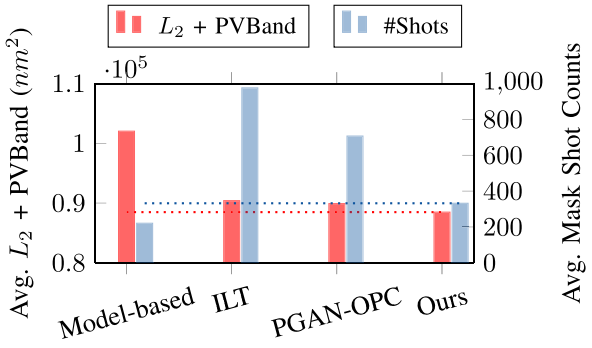


Fig. 11. Performance on average mask fracturing shots count and average L_2 + PVBand score of different methods.

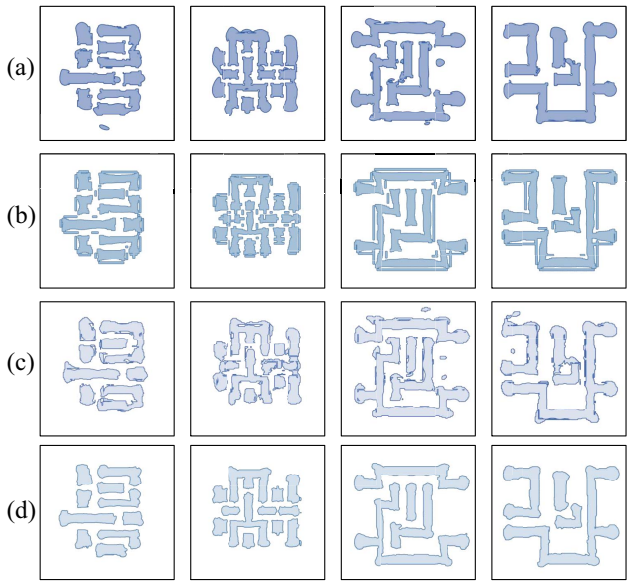


Fig. 12. Mask complexity visualizations of (a) conventional ILT [4], (b) model-based OPC [1], (c) GAN-OPC family [28], and (d) Neural-ILT family.

generate clean mask solutions with lower complexity comparing with ILT and GAN-OPC family, while achieving the best mask printability [L_2+PVB (nm^2)] and 23×∼30× TAT speedup. Note that model-based OPC achieves the lowest mask complexity among all methods since its segmentwise modifications will only generate rectilinear shapes. However, such limited searching space sacrifices the solution quality, which leads to the worst mask printability [L_2+PVB (nm^2)] among all methods in Fig. 11.

The above results have demonstrated the effectiveness and superiority of our Neural-ILT framework, which is able to achieve very significant TAT speedup with better mask manufacturability and SOTA mask printability. It is worth mentioning that the significant speedup of Neural-ILT is mainly contributed from three aspects: 1) the acceleration by the CUDA-based litho-simulator; 2) the better initial solution predicted by the domain-specific pretrained backbone model; and 3) higher searching efficiency of `PyTorch` built-in numerical optimizer.

B. Evaluation of Multitask-Enabled Neural-ILT

To verify the feasibility and effectiveness of the multitask-enabled Neural-ILT, we pretrain a backbone model $\mathbf{w}_{\text{pt}}$ for the Neural-ILT-multitask following the steps described in Section III-F, and test it on the NanGate FreePDK45 library [25] for 45-nm cut-layer EPE minimization task. Note that the layout decomposition results of the FreePDK45 dataset is provided by Zhong *et al.* [24], and we follow exactly the same experimental settings in [24], where the EPEV threshold th_{epe} is set to 10 nm, $\theta_M^{45\text{nm}} = 8$, $\theta_Z^{45\text{nm}} = 120$, and $I_{th}^{45\text{nm}} = 0.039$.

For each input testcase, we use identical Neural-ILT configurations, where the maximum ILT iterations = 70, initial learning rate lr = 0.002, lr decay rate = 0.1, lr decay step size = 40 iterations, early stop patience = 15 iterations. According to the performance profiles in Fig. 9, we set $\delta = 2$ for $L_{\text{epe\_fast}}$ (14) to achieve the best tradeoff.

As shown in Table I, Neural-ILT-multitask [denoted as "Ours (Multitask)"] is capable of handling the new task with a very competitive performance. It outperforms the SOTA work [24] (denoted as "TCAD'21") with 13% less average L_2 error, 25% less average EPEVs, and 7.6× runtime speedup.

In conclusion, we successfully leverage the off-the-shelf Neural-ILT utility to achieve rapid task adaptation from the domain of 32-nm metal-layer to the domain of 45-nm cut-layer, which verify the feasibility and development efficiency of our multitask-enabled Neural-ILT prototype.

C. On the Scalability of Neural-ILT Family

We further examine the scalability of the Neural-ILT family by conducting experiments on ten additional testcases (cases 11–20 in Table III) that contain more patterns and larger target pattern areas than the original IBM benchmarks (cases 1–10 in Table I).

We use identical on-neural-network ILT correction configurations as in the previous evaluation of Section IV-A, quantitative results are listed in Table III.[2] It is notable that our framework outperforms the conventional methods in terms of all metrics. More specifically, comparing with ILT and PGAN-OPC, we achieve 26.4% and 21.1% squared L_2 error reductions; 1.5% and 2% PVBand reduction; and more than 27× and 21× turnaround runtime speedup. In conclusion, Neural-ILT demonstrates clear superiority over the conventional methods on larger benchmarks, which indicates better scalability of our framework.

D. On the Necessity of Domain-Specific Pretrain

In Section III-E, we proposed a domain-specific training recipe to pretrain the backbone model of Neural-ILT, where a refined dataset $\mathcal{D}_{\text{refine}}$ is generated based on the original dataset $\mathcal{D}$. Then, the domain knowledge of the partial coherent lithography model is explicitly introduced by penalizing the ILT loss in (12) during the network pretraining.

The necessity of the above domain-specific training recipe is twofold. First, this training strategy strengthens the potentials

[2]Note that the results of model-based OPC are not presented in Table III as the binary released in [1] crashed on the new benchmarks.

TABLE III
PERFORMANCE COMPARISONS† ON LARGER BENCHMARKS

Case	ILT-based* [4]			PGAN-OPC* [18]			Ours				
	L2	PVB	TAT	L2	PVB	TAT	L2	PVB	#EPE	#shots	TAT
11	94792	125578	-	89229	121276	-	79933	120577	12	669	20
12	94604	128306	-	93454	127551	-	86995	104266	15	556	12
13	136861	160327	-	134397	155511	-	133281	152718	70	766	15
14	69090	79337	-	58776	85644	-	43797	92137	0	455	14
15	96961	116039	-	99830	116728	-	69521	122115	3	808	19
16	98159	115107	-	96335	113981	-	73790	117359	2	764	19
17	79192	91989	-	71522	94841	-	49031	92320	0	531	19
18	65572	81503	-	60372	83718	-	47409	84971	0	478	16
19	107095	121922	-	105973	122770	-	93922	115028	5	614	14
20	62537	78319	-	57086	81285	-	38028	80127	0	452	19
Avg.	90486.3	109842.7	455.0	86697.4	110330.5	364.0	**71570.7**	**108161.8**	10.7	609.3	**16.8**
Ratio	1.00	1.00	1.00	0.96	1.00	0.80	**0.79**	**0.98**	-	-	**0.04**

* Results quoted from [28], #EPE, #shots and detail TAT of ILT [4] and PGAN-OPC [18] are not released.
† Model-based method's results [1] are not available since its binary crashed on the larger benchmarks.
Unit: L_2 (nm^2), PVB (nm^2), TAT (s).

of Neural-ILT in terms of both prediction quality and searching efficiency. To verify this claim, in Fig. 13, we profiled the loss curves of on-neural-network ILT corrections based on two different pretrained backbone models: 1) a regular backbone model pretrained without DSR, i.e., trained with dataset $\mathcal{D}$ using (3) and 2) a domain-specific backbone model pretrained with DSR, i.e., trained with dataset $\mathcal{D}_{\text{refine}}$ using (12). We use identical Neural-ILT configurations as in Section IV-A, but lager maximum ILT iterations $= 70$. The loss curves of the regular backbone model are marked in blue, while that of the domain-specific backbone model are marked red. The solid and dotted lines represent the L_2 loss and the printability score (L_2+PVBand), respectively. As depicted in Fig. 13, the domain-specific backbone model Neural-ILT (red curves) outperforms the regular backbone model Neural-ILT (blue curves) in terms of: 1) faster convergence speed, most red curves reach convergence within 20 iterations, while the blue curves usually take around $30 \sim 40$ iterations; 2) lowest L_2 loss attainable; and 3) smoother loss curves, where the blue curves need to jump out of local minima frequently, thereby creating numbers of zigzags which hinder the ILT optimization in general. The above three points indicate better searching quality and efficiency of our domain-specific backbone model.

Second, the DSR training strategy helps Neural-ILT to achieve SOTA ILT correction quality on a simple and lightweight model (e.g., a standard UNet), thereby avoiding unnecessary overhead on over-complicated network architecture and computational resource occupancy. The entire Neural-ILT contains only 7.8M trainable parameters, which takes up only 6.1% of the best GAN-OPC framework (7.8M versus 127M.)[3]

E. On the Necessity of On-Neural-Network ILT Correction

Here, we want to have a short discussion on what really distinguishes Neural-ILT (on-neural-network ILT) from general ILT (on-mask-image ILT). We designed an experiment for both conventional ILT and Neural-ILT on five additional testcases from the test dataset (blind for model training). In order to ensure a fair comparison, we develop GPU-ILT, a GPU version of the conventional ILT tool [4] using our CUDA-based litho-simulation tool. For each testcase, Neural-ILT performs 40 iterations of corrections with initial learning rate $\text{lr} = 0.001$, while GPU-ILT performs 100 iterations of corrections with initial learning rate $\text{lr} = 1$. Quantitative results are depicted in Table IV. It is notable that comparing to on-mask ILT (GPU-ILT), Neural-ILT consumes less ILT iterations (i.e., 100 versus 40) with much smaller initial learning rate (i.e., 1.0 versus 0.001), while achieving better overall quality (i.e., 9% better printability, 51% less mask shots counts).

TABLE IV
MASK PRINTABILITY, COMPLEXITY, AND RUNTIME PERFORMANCE COMPARISON BETWEEN GPU-ILT AND NEURAL-ILT

	GPU-ILT			Ours		
Case	TAT (s)	Score*	# shots	TAT (s)	Score*	# shots
A	18.81	128005	1636	11.93	113357	591
B	18.78	90922	1025	11.12	85568	610
C	18.77	101815	1158	11.01	93874	652
D	18.89	118648	1193	14.72	108631	620
E	18.78	120501	1279	11.16	105202	577
Avg.	18.8	111978.2	1258.2	12.0	101326.4	610.0
Ratio	1.00	1.00	1.00	**0.64**	**0.91**	**0.49**

* Score $= L_2$ $(nm^2) +$ PVB (nm^2).

Recall that general ILT treats the mask optimization as an inverse problem of the lithography imaging system [function $f(\cdot;\cdot)$], which essentially tries to find f^{-1}. Considering it is ill-posed, conventional ILT cannot directly model f^{-1} and hence need to modify the mask iteratively based on gradient descent. However, Neural-ILT is built on top of a neural network, which allows smooth and fine-grained refinement on the weights and neurons' activities with considerably larger searching space. With the powerful capability of neural network in function approximation, the feedforward computation for mask generation $\phi(\mathbf{Z}_t; \hat{\mathbf{w}})$ in Neural-ILT essentially serves as such an approximated inverse lithography function $f^{-1}(\mathbf{Z}_t; \mathbf{P}_{\text{nom}})$ and results in efficient computation.

[3]Quoted from [28], EGAN-OPC contains 127M trainable parameters, 39M for the generator, 88M for the discriminator.

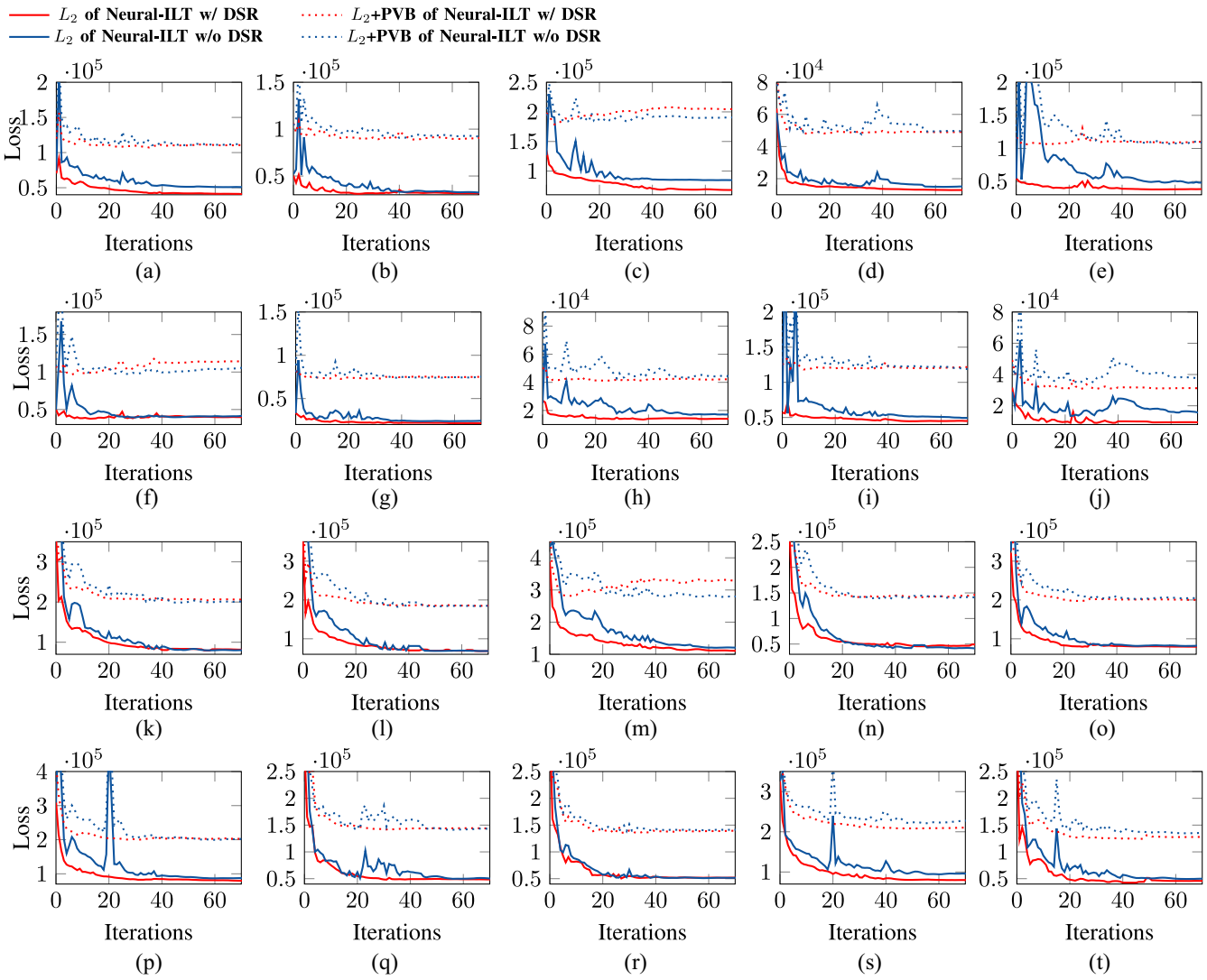


Fig. 13. Loss curves (solid line: L_2; dotted line: L_2 + PVBand) of on-neural-network ILT corrections based on two different pretrained backbone models: 1) normal backbone model pretrained without DSR, i.e., trained with the original dataset $\mathcal{D}$ using (3), marked in blue; 2) domain-specific backbone model pretrained with DSR, i.e., trained with the refined dataset $\mathcal{D}_{\text{refine}}$ using (12), marked in red. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Case 5. (f) Case 6. (g) Case 7. (h) Case 8. (i) Case 9. (j) Case 10. (k) Case 11. (l) Case 12. (m) Case 13. (n) Case 14. (o) Case 15. (p) Case 16. (q) Case 17. (r) Case 18. (s) Case 19. (t) Case 20.

Moreover, a neural network can be pretrained in advance, which offers a better initial solution for achieving faster ILT convergence.

In conclusion, we believe that a properly pretrained neural network is more suitable for performing ILT-style mask correction benefiting from the potentially faster and better ILT convergence. The potential of transferability of Neural-ILT can be further explored leveraging the real industrial lithography recipes and designs in advanced technology nodes.

V. Conclusion

In this article, we proposed Neural-ILT, an end-to-end learning-based OPC framework that literally conducts on-neural-network ILT for the given layouts. We believe that the proposed paradigm can be extended on industrial applications once equipped with industrial ILT solver and lithography recipe. Future works would include further studies on the applications of Neural-ILT.

References

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

[2] 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.

[3] F. Liu and X. Shi, "An efficient mask optimization method based on homotopy continuation technique," in *Proc. IEEE/ACM Design Autom. Test Europe (DATE)*, 2011, pp. 1–6.

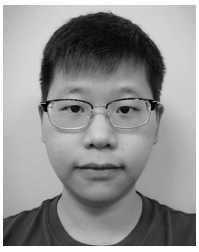
[4] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[5] K. Hooker, B. Kuechler, A. Kazarian, G. Xiao, and K. Lucas, "ILT optimization of EUV masks for sub-7nm lithography," in *Proc. SPIE*, vol. 10446, 2017, Art. no. 1044604.

[6] 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)*, 2017, pp. 81–88.

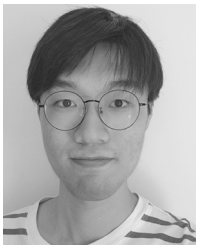
[7] R. Pearman *et al.*, "How curvilinear mask patterning will enhance the EUV process window: A study using rigorous wafer+ mask dual simulation," in *Proc. SPIE*, vol. 11178, 2019, Art. no. 1117809.

[8] I. Torunoglu *et al.*, "A GPU-based full-chip inverse lithography solution for random patterns," in *Proc. SPIE*, 2010, Art. no. 764115.
[9] V. Domnenko *et al.*, "EUV computational lithography using accelerated topographic mask simulation," in *Proc. SPIE*, 2019, Art. no. 109620O.
[10] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2013, pp. 1–6.
[11] J. Kuang, J. Ye, and E. F. Y. Young, "Simultaneous template optimization and mask assignment for DSA with multiple patterning," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2016, pp. 75–82.
[12] 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.
[13] Y. Lin *et al.*, "Data efficient lithography modeling with transfer learning and active data selection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 10, pp. 1900–1913, Oct. 2019.
[14] D. Ding, B. Yu, J. Ghosh, and D. Z. Pan, "EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2012, pp. 263–270.
[15] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1175–1187, Jun. 2019.
[16] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
[17] X. Ma, S. Jiang, J. Wang, B. Wu, Z. Song, and Y. Li, "A fast and manufacture-friendly optical proximity correction based on machine learning," *Microelectron. Eng.*, vol. 168, pp. 15–26, Jan. 2017.
[18] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2018, p. 131.
[19] 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.
[20] T. Chan, P. Gupta, K. Han, A. A. Kagalwalla, and A. B. Kahng, "Benchmarking of mask fracturing heuristics," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 1, pp. 170–183, Jan. 2017.
[21] C. Mack, *Fundamental Principles of Optical Lithography: The Science of Microfabrication*. Hoboken, NJ, USA: Wiley, 2008.
[22] 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.
[23] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.
[24] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Feb. 23, 2021, doi: 10.1109/TCAD.2021.3061494.
[25] (2008). *NanGate FreePDK45 Generic Open Cell Library*. [Online]. Available: http://www.si2.org/openeda.si2.org/projects/nangatelib
[26] A. Poonawala and P. Milanfar, "OPC and PSM design using inverse lithography: A nonlinear optimization approach," in *Proc. Opt. Microlithogr. XIX*, vol. 6154, 2006, Art. no. 61543H.
[27] B. Jiang *et al.*, "Fit: Fill insertion considering timing," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 221.
[28] 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.



Bentian Jiang received the B.Eng. degree in electronics and information engineering from Sichuan University, Chengdu, China, in 2017, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2021.

His research interests include very large-scale integration design for manufacturability and physical design.



Lixin Liu received the B.Eng. degree in electronic science and technology from the South China University of Technology, Guangzhou, 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 research interest includes deep learning and its applications on physical design.

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. He is currently with The Chinese University of Hong Kong, Hong Kong. His research interest includes very large-scale integration 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.

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 the 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.

Evangeline F. Y. Young (Senior Member, IEEE) received the B.Sc. degree in computer science from The Chinese University of Hong Kong (CUHK), Hong Kong, and the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 1999.

She is currently a Professor with the Department of Computer Science and Engineering, CUHK. Her research focuses on floorplanning, placement, routing, DFM, and EDA on physical design in general. Her research interests include EDA, optimization, algorithms, and AI.

Prof. Young's research group has won the best paper awards from ICCAD 2017, ISPD 2017, SLIP 2017, and FCCM 2018, and several championships and prizes in renown EDA contests, including the 2018–2020, 2015–2016, 2012–2013 CAD Contests at ICCAD, DAC 2012, and ISPD 2015–2020 and 2010–2011. She has served on the organization committees of ICCAD, ISPD, ARC, and FPT and on the program committees of conferences, including DAC, ICCAD, ISPD, ASP-DAC, SLIP, DATE, and GLSVLSI. She also served on the editorial boards of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *ACM Transactions on Design Automation of Electronic Systems*, and *Integration, the VLSI Journal*.