Neural-ILT: Migrating ILT to Neural Networks for Mask Printability and Complexity Co-optimization

Bentian Jiang
Chinese University of Hong Kong
btjiang@cse.cuhk.edu.hk

Lixin Liu
Chinese University of Hong Kong
lxliu@cse.cuhk.edu.hk

Yuzhe Ma
Chinese University of Hong Kong
yzma@cse.cuhk.edu.hk

Hang Zhang
Cornell University
hz459@cornell.edu

Bei Yu
Chinese University of Hong Kong
byu@cse.cuhk.edu.hk

Evangeline F.Y. Young
Chinese University of Hong Kong
fyyoung@cse.cuhk.edu.hk

ABSTRACT

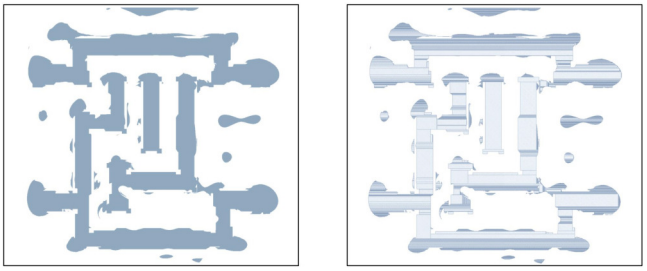
Optical proximity correction (OPC) for advanced technology node now has become extremely expensive and challenging. Conventional model-based OPC encounters performance degradation and large process variation, while aggressive approach such as inverse lithography technology (ILT) suffers from large computational overhead for both mask optimization and mask writing processes. In this paper, 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. Quantitative results show that, comparing to the state-of-the-art (SOTA) learning-based OPC solution and conventional ILT flow, Neural-ILT can achieve **30×** ∼ **70×** turn around time (TAT) speedup with lower mask complexity and comparable mask printability. We believe this work could arouse the interests of bridging well-developed deep learning toolkits to GPU-based high-performance lithographic computations to achieve groundbreaking performance boosting on various computational lithography-related tasks.

1 INTRODUCTION

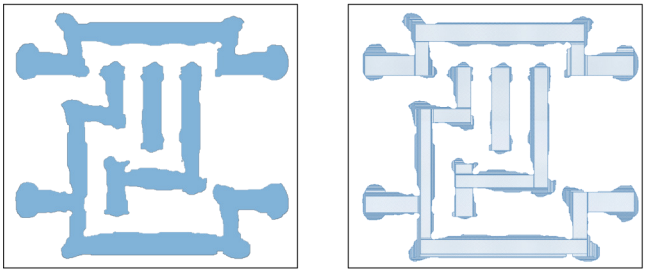
Computational lithography models are designed to learn the printing effects of real lithography patterns. Building on top of these delicate lithographic models, advance resolution enhancement techniques (RETs) such as sub-resolution assist feature (SRAF) insertion and optical proximity correction (OPC) help the designers to obtain optimized masks that result in high fidelity printed patterns [1].

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. Conventional OPC approaches can be roughly categorized into (1) model-based OPC [2, 3] and (2) inverse lithography technology-based (ILT) OPC [4–7]. 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, inverse lithography technology uses numerical approach, which treats OPC as an inverse imaging problem, to perform pixel-wise mask optimization. Nowadays, ILT is widely adopted to find flexible 193i and even EUV mask pattern solutions to improve overall process window [6, 8]. 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



(a) Conventional ILT [5] synthesized mask (high complexity) and corresponding mask fracturing result with 2045 shots.



(b) Neural-ILT synthesized mask (low complexity) and corresponding mask fracturing result with only 653 shots.

Figure 1: Visualizations of mask fracturing results on the synthesized masks with different complexities.

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415704

have been made, including but not limited to (1) migrating high-performance computational lithography to GPU acceleration [9]; (2) introducing fast modeling approaches for rigorous/compact litho-simulations [10]; (3) considering multiple patterning [11, 12] and (4) applying the SOTA machine learning techniques on lithography-related applications such as lithography system modeling [13, 14], hotspot detection [15–17] and OPC [13, 18–20]. Among them, Yang *et al.* [19] (GAN-OPC) for the first time applied conditional generative adversarial networks (CGAN) 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 [5] for further refinement, which helps to achieve faster convergence and better mask printability. Ye *et al.* [13] (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 trade off between result quality and turn around time. For supervised learning, the prediction quality is highly related to the quality of the training datasets, and 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 (e.g., Figure 1). The ideal curvilinear shapes (Figure 1(a)) generated by conventional ILT [5] require huge amounts of shots to accurately replicate the shapes, which leads to extremely poor manufacturability in high-volumes due to the unmanageable mask writing times [8, 21]. Observing the fact that reducing the mask complexity can significantly reduce the shot count (2045 in Figure 1(a) vs. 653 in Figure 1(b)), it is imperative to consider mask complexity as an optimization objective during the ILT correction.

Motivated by these issues, a new challenge naturally arises: can we completely replace the conventional ILT-based OPC flow by **purely** machine learning-based solutions, so as to simultaneously achieve breakthrough in runtime boosting and the SOTA result quality, while considering mask manufacturability? In this work, we develop Neural-ILT, an end-to-end high-performance ILT-based OPC framework in a single neural network. This work attempts to completely replace the conventional end-to-end ILT process (based on a partial coherent imaging system) with purely learning-based techniques. Unlike previous learning-based OPC solutions [19, 20], Neural-ILT is able to directly generate the masks after OPC for the unseen test layouts without additional rigorous refinement on the network output. Our key contributions are summarized as follows.

- 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.
- A special training recipe with domain knowledge of a partial coherent lithography imaging system is applied to pre-train the backbone network of Neural-ILT in a supervised manner.
- 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.
- Experimental results show that Neural-ILT achieves breakthrough turn around time speedup with lower mask complexity and high mask printability comparable to the SOTA.

The rest of the paper is organized as follows. Section 2 lists some preliminaries. Section 3 discusses the details of the framework and algorithms. Section 4 presents experimental results, followed by a conclusion in Section 5.

2 PRELIMINARIES

In this section, we will go through the background and problem formulation. Throughout the rest of the paper, we denote $\mathbf{Z}_t$ as the target layout, $\mathbf{M}$ as the mask, $\mathbf{I}$ as the intensity (aerial) image, $\mathbf{Z}$, $\mathbf{Z}_{\text{in}}$, $\mathbf{Z}_{\text{out}}$ 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 element-wise product, and $\phi(\cdot;\cdot)$ stands for the forward function of the neural network.

2.1 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 partially coherence imaging system [22] is widely adopted to approximate the printing behavior. The lithography model produces an 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 optical kernels $\mathbf{H}$ which represent the singular value decomposition of the optical system (193nm wavelength system with annular illumination in this paper):

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

where $\mathbf{h}_k$ is the k^{th} kernel of the model and ω_k is the corresponding weight. The system can be further simplified by an N_h^{th} order approximation [5], where $N_h = 24$ in our implementation. A resist model which 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:

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

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

2.2 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 Figure 1), which makes mask writing extremely expensive [8]. Despite the fact that more advanced mask data preparation techniques like multi-beam fracturing and model-based mask data preparation (MB-MDP) have been deployed [8], in this paper, 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, which are defined as follows:

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

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

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

3 THE NEURAL-ILT ALGORITHMS

The objective of a typical end-to-end ILT OPC process is to find a mask solution $\mathbf{M}_{opt} = f^{-1}(\mathbf{Z}_t; \mathbf{P}_{nom})$ for the given layout $\mathbf{Z}_t$, where $f(\cdot; \mathbf{P}_{nom})$ is the forward lithography simulation under nominal condition. Since different mask solutions may yield the same wafer image, there is no explicit closed-form formula for solving the inverse lithography process $f^{-1}(\cdot; \mathbf{P}_{nom})$. 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, for the purpose of mimicking the end-to-end ILT correction process, a specially designed neural network architecture called auto-encoder can be applied to pixel-wise mask prediction.

The overall flow of Neural-ILT is illustrated in Figure 2. Neural-ILT consists of three components: (1) a pre-trained backbone UNet, (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 pre-trained 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 fine-grained mask through the forward inference.

Algorithm 1 CUDA-based Lithography Simulation

Input: Input mask $\mathbf{M}$, lithography 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;
  $\mathbf{M}_{cplx} \leftarrow$ Initialize each pixel $\mathbf{M}_{cplx}(x, y)$ as complex value;
  $\mathbf{M}_{cplx}$.real $\leftarrow \mathbf{M} *$ Dose, $\mathbf{M}_{cplx}$.img $\leftarrow \mathbf{0}$;
  $\mathbf{M}_{fft} \leftarrow$ CUDA_FFT($\mathbf{M}_{cplx}$);
  $\mathbf{I}_{fft} \leftarrow$ Initialize each pixel $\mathbf{I}_{fft}(x, y)$ as a complex vector;
  for k=1,..,24 **do**
    Pad $\mathbf{H}_k$ with 0 to the size of $\mathbf{M}_{fft}$;
    $\mathbf{I}_{k\_fft}$.real $\leftarrow \mathbf{M}_{fft}$.real $\odot \mathbf{H}_k$.real $- \mathbf{M}_{fft}$.img $\odot \mathbf{H}_k$.img;
    $\mathbf{I}_{k\_fft}$.img $\leftarrow \mathbf{M}_{fft}$.img $\odot \mathbf{H}_k$.real $+ \mathbf{M}_{fft}$.real $\odot \mathbf{H}_k$.img;
  $\mathbf{I}_{ifft} \leftarrow$ CUDA_IFFT($\mathbf{I}_{fft}$);
  if Mode == Convolve **then** ▷ Litho-convolution
    for each pixel $\mathbf{I}_{ifft}(x, y)$ **do**
      $\mathbf{I}_{sqrt}(x, y) \leftarrow \sum_{k=1}^{24} \boldsymbol{\omega}_k^{\frac{1}{2}} * \mathbf{I}_{k\_ifft}(x, y)$;
    return Square rooted intensity map $\mathbf{I}_{sqrt}$;
  if Mode == Simulation **then** ▷ Litho-simulation
    for each pixel $\mathbf{I}_{ifft}(x, y)$ **do**
      $\mathbf{I}(x, y) \leftarrow \sum_{k=1}^{24} \boldsymbol{\omega}_k * \mathbf{I}_{k\_ifft}^2(x, y)$;
    $\mathbf{Z} \leftarrow$ Apply resist model in Equation (2) on $\mathbf{I}$;
    return Intensity map $\mathbf{I}$, wafer image $\mathbf{Z}$;

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

3.1 CUDA-based High-performance Lithography Simulation

As mentioned in Section 3, 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 model 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 maximizing the algorithmic parallelism and hardware resource utilization, we develop a GPU-based high performance lithography simulation tool with CUDA based on the ICCAD

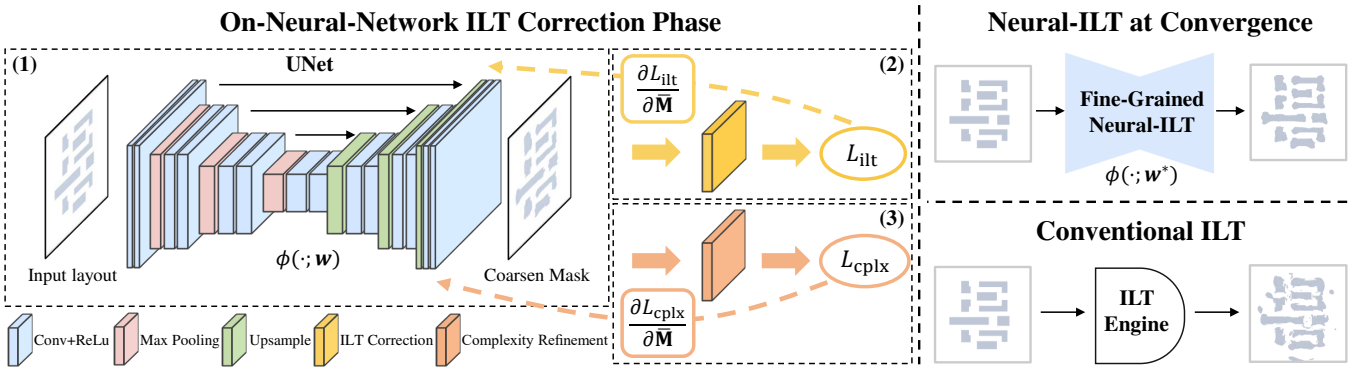


Figure 2: Overview of Neural-ILT. The on-neural-networks ILT correction is equivalent to the training procedure of Neural-ILT. At convergence, the fined-grained Neural-ILT is able to directly generate optimized mask with reasonably good printability and low complexity.

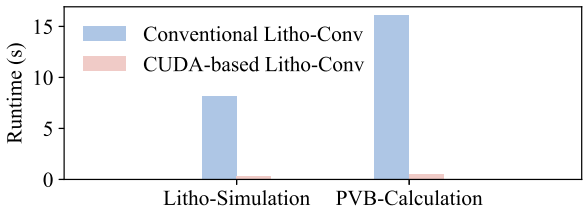


Figure 3: Runtime comparisons for lithography simulation and PVBand calculation.

2013 contest lithography evaluation tool [23]. 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.

As shown in Figure 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 platform like `PyTorch` or `TensorFlow`. In the following subsections, we will show how the proposed CUDA-based litho-simulation tool helps to drive the key optimization process of Neural-ILT.

3.2 Pre-trained Backbone Neural Network

Finding a mapping between an input layout and a output mask is usually cast as an image-to-image translation task. In our case, a general type of neural network called auto-encoder can be applied to make pixel-wise binary classification on whether a pixel belongs to the optimized mask. A standard auto-encoder architecture is composed of two sub-networks: (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 [13, 19], inside Neural-ILT, a pre-trained backbone model based on the auto-encoder structure is required to provide the basic layout-to-mask translation

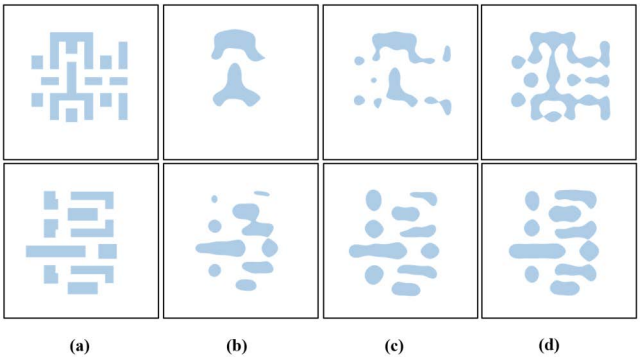


Figure 4: (a) Target layouts. Wafer images generated by (b) layouts, (c) UNet direct outputs, (d) ILT synthesized masks.

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 auto-encoder-like networks, rather than dependent on the use of fancy network architectures. Based on such consideration, we selected to use UNet [24], a well-known but relatively simple model which is similar to auto-encoder, to serve as the basic module of Neural-ILT. As depicted in Figure 2, the UNet architecture consists of a down-sampling path to capture context and a symmetric up-sampling path that enables pixel-wise mask correction (please refer to [24] for details). Given a set of input target layouts $\mathcal{Z}_t = \{\mathbf{Z}_{t,1}, \mathbf{Z}_{t,2}, \ldots, \mathbf{Z}_{t,n}\}$ and a corresponding optimized mask set $\mathcal{M}^* = \{\mathbf{M^*}_1, \mathbf{M^*}_2, \ldots, \mathbf{M^*}_n\}$, the training procedure of the UNet is to minimize the following objective:

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

where $\phi(\cdot; \mathbf{w})$ is the network forward output for the given model weights $\mathbf{w}$, and λ is a configurable hyper-parameter. Figure 4 shows the image fidelity comparison between the wafer images generated by layouts, pre-trained UNet outputs, and ILT synthesized masks. We can see that, the pre-trained UNet does improve the mask printability (corresponds to better wafer image fidelity) to some extent

Algorithm 2 ILT Correction Layer Forward and Backward

Input: Masks $\mathbf{M}$, $\bar{\mathbf{M}}$, target 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);
  $L_{\text{ilt}} \leftarrow ||\mathbf{Z} - \mathbf{Z}_t||_2^{\gamma}$; ▷ $\gamma = 4$ in forward
  return Lithography loss L_{ilt};
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 (5) using Algorithm 1
  return Gradient $\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}}$;

comparing with the wafer images yielded by the original layouts. However, when dealing with complex layouts, the printability of UNet outputs can hardly be maintained satisfactorily, especially in comparison with 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 lithography error. Such demands indicate necessity and rationality of our on-neural-network ILT correction paradigm.

3.3 ILT Correction and Mask Complexity Refinement Layers

As highlighted in Section 1, 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.

3.3.1 ILT Correction Layer. The objective of conventional 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 corresponding wafer image of $\mathbf{M}$; N denotes the image dimension, and γ is a configurable parameter. In order to obtain the gradient $\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}}$ for updating the mask, $\mathbf{Z}$ and $\mathbf{M}$ should be regarded as matrices with continuous values so as to make L_{ilt} differentiable. The binary constraints is commonly relaxed by sigmoid function so that the variables become unconstrained, and here we introduce new 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 replaced with their sigmoid approximations

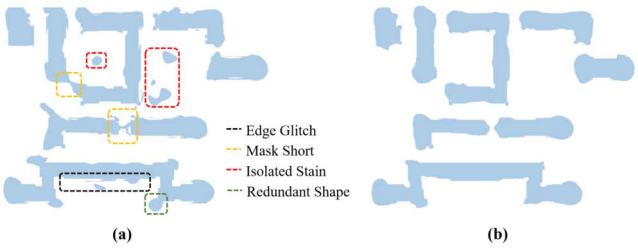


Figure 5: Mask complexity comparison: (a) ILT synthesized mask with high complexity and (b) mask purified by complexity refinement.

$\mathbf{M} = \text{sig}(\bar{\mathbf{M}}) = \frac{1}{1+\exp(-\theta_M \times \bar{\mathbf{M}})}$ and $\mathbf{Z} = \text{sig}(\mathbf{I}) = \frac{1}{1+\exp(-\theta_Z \times (\mathbf{I} - I_{th}))}$, where θ_M and θ_Z define the steepness of the sigmoid functions used for $\mathbf{M}$ and $\mathbf{Z}$, respectively. Following the derivations in [25], the gradient of the above lithography loss is given by

$$\begin{aligned}\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}} =& \gamma \times (\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \frac{\partial \mathbf{Z}}{\partial \mathbf{M}} \odot \frac{\partial \mathbf{M}}{\partial \bar{\mathbf{M}}} \\ =& \gamma\theta_M\theta_Z \times \{\mathbf{H}^{\text{flip}} \otimes [(\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z}) \odot (\mathbf{M} \otimes \mathbf{H}^*)] \\ &+ (\mathbf{H}^{\text{flip}})^* \otimes [(\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z}) \odot (\mathbf{M} \otimes \mathbf{H})]\} \\ &\odot \mathbf{M} \odot (1 - \mathbf{M}),\end{aligned} \tag{5}$$

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

Considering that the loss and gradient computations of conventional ILT correction share similar mechanisms of neural network forward and backward propagation, conventional ILT functionality can be implemented in customized ILT correction layer on neural network. Regarding L_{ilt} as the forward propagation loss, and $\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}}$ as the backward propagation gradient, the weights of the predecessor neural networks $\mathbf{w}_{\text{net}}$ can be updated through the chain rule $\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \bar{\mathbf{M}}} \frac{\partial \bar{\mathbf{M}}}{\partial \phi(\mathbf{Z}_t; \mathbf{w}_{\text{net}})} \frac{\partial \phi(\mathbf{Z}_t; \mathbf{w}_{\text{net}})}{\partial \mathbf{w}_{\text{net}}}$. 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` or `TensorFlow` to fully leverage its computational efficiency. Algorithm 2 depicts how to implement the forward and backward computations of the ILT correction layer with Algorithm 1.

3.3.2 Mask Complexity Refinement Layer. Masks synthesized by ILT usually consist of non-rectangular 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 Figure 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 non-manufacturing-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 (Figure 5(a)). They usually will not be printed on wafer image under the *min process* condition

$\mathbf{P}_{\min}$, but are likely to be printed at the *nominal process* condition $\mathbf{P}_{\text{nom}}$ and 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}:

$$L_{\text{cplx}} = ||\mathbf{Z}_{\text{in}} - \mathbf{Z}_{\text{out}}||_2^2. \tag{6}$$

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

$$\frac{\partial L_{\text{cplx}}}{\partial \bar{\mathbf{M}}} = 2 \times (\mathbf{Z}_{\text{in}} - \mathbf{Z}_{\text{out}}) \odot (\mathbf{Z}_{\text{in}}{}' - \mathbf{Z}_{\text{out}}{}'). \tag{7}$$

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

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

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 implementations of the layer forward and backward computations (similar to Algorithm 2) are omitted here due to the limited space. The effectiveness of the mask complexity refinement layer are demonstrated in Figure 5. Note that, from the perspective of deployment, we should always connect the complexity refinement layer with an ILT correction layer to form a multi-objective loss function (Equation (8)). Otherwise, optimization with only mask complexity refinement layer will eliminate all mask shapes which literally yield "0" loss for Equation (6).

3.4 Neural-ILT for Mask Printability and Complexity Co-optimization

As presented in Figure 2, the Neural-ILT network model is composed of 3 modules: (1) a pre-trained 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 following objective,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}} \overbrace{\alpha \, ||f(\phi(\mathbf{Z}_{\text{t}}; \mathbf{w}); \mathbf{P}_{\text{nom}}) - \mathbf{Z}_{\text{t}}||_\gamma^\gamma}^{L_{\text{ilt}}} + \underbrace{\beta \, ||f(\phi(\mathbf{Z}_{\text{t}}; \mathbf{w}); \mathbf{P}_{\min}) - f(\phi(\mathbf{Z}_{\text{t}}; \mathbf{w}); \mathbf{P}_{\max})||_2^2}_{L_{\text{cplx}}}, \tag{8}$$

where α, β and γ are configurable hyper-parameters.

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 weights $\mathbf{w}$ of $\phi(\cdot; \mathbf{w})$ by backward propagation through the chain rule:

$$\frac{\partial L_{\text{refine}}}{\partial \mathbf{w}} = \frac{\partial L_{\text{refine}}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \bar{\mathbf{M}}} \frac{\partial \bar{\mathbf{M}}}{\partial \phi(\mathbf{Z}_{\text{t}}; \mathbf{w})} \frac{\partial \phi(\mathbf{Z}_{\text{t}}; \mathbf{w})}{\partial \mathbf{w}}, \tag{9}$$

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

3.5 Retrain Backbone with Domain Knowledge of Partial Coherent Imaging System

The original training dataset is composed of 4300 pairs of target layout and reference mask, and the reference masks are synthesized by a conventional ILT tool [5]. However, as shown in Figure 5(a), ILT synthesized masks usually consist of numerous complex features. In order to improve the training data quality, a Neural-ILT model, which contains a UNet trained by the original dataset, is used to perform mask complexity refinement for all the original training instances. These output masks (e.g. Figure 5(b)) together with all the original target layouts (thus a total of 8600) form a newly refined dataset. We can use this refined dataset to re-train the UNet with the following cycle loss L_{cycle}:

$$L_{\text{cycle}} = ||\phi(\mathcal{Z}_{\text{t}}; \mathbf{w}) - \mathcal{M}^*||_2^2 + \eta ||f(\phi(\mathcal{Z}_{\text{t}}; \mathbf{w}); \mathbf{P}_{\text{nom}}) - \mathcal{Z}_{\text{t}}||_2^2, \tag{10}$$

where $\mathcal{Z}_{\text{t}}$ and $\mathcal{M}^*$ refer to the input target layouts and reference ILT mask labels in the refined dataset; η is a configurable hyper-parameter. The calculations of the first and second terms in L_{cycle} form a close-loop. The first term of Equation (10) minimizes the image difference between network predictions and labels. The second term is essentially the ILT loss in Equation (8), 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 second term in Equation (10) serves as a regularization term, which guides the re-trained network $\phi(\cdot\,;\, \mathbf{w})$ gradually converged along a domain-specified direction. The explicit introduction of the partially coherent lithography system in network training is also a key contribution that sets us apart from the previous learning-based OPC works.

Benefit from above refinements, our Neural-ILT with the re-trained UNet is able to achieve faster convergence with better mask printability and lower complexity.

4 EXPERIMENTAL RESULTS

The Neural-ILT framework is developed with `PyTorch` and `CUDA`, and tested on a Linux machine with 2.2GHz Intel Xeon CPU and a single Nvidia Titan V GPU. The original training dataset is obtained from the authors of GAN-OPC [19], which synthesizes over 4300 training instances based on the design specifications from existing 32*nm* M1 layout topologies. The UNet is pre-trained for 20 epochs which takes around 19 hours on a single Titan V GPU. As for the evaluation, the lithography recipe is provided by ICCAD 2013 contest evaluation package [23] for 32*nm* M1 layout designs, and the mask fracturing tool is implemented based on an efficient contour decomposition algorithm in work [26]. Note that, to the best of our knowledge, the lithography engine and benchmarks provided in ICCAD 2013 contest is the only open-sourced industrial package (with 32*nm* or below technology node) available for public research. Considering such limitation, we can only evaluate Neural-ILT leveraging the ICCAD13 contest evaluation suite [23].

4.1 The Effectiveness of Neural-ILT

To verify the effectiveness of the proposed Neural-ILT, 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

Table 1: Mask Printability, Complexity and Runtime Performance Comparison with SOTA Methods

Benchmarks		ILT [5]				PGAN-OPC [19]				Neural-ILT†			
ID	Area (nm^2)	TAT (s)	L_2 (nm^2)	PVB (nm^2)	# shots	TAT (s)	L_2 (nm^2)	PVB (nm^2)	# shots	TAT (s)	L_2 (nm^2)	PVB (nm^2)	# shots
case1	215344	1280	49893	65534	2478	358	52570	56267	931	13.57	50795	63695	743
case2	169280	381	50369	48230	704	368	42253	50822	692	14.37	36969	60232	571
case3	213504	1123	81007	108608	2319	368	83663	94498	1048	9.72	94447	85358	791
case4	82560	1271	20044	28285	1165	377	19965	28957	386	10.40	17420	32287	209
case5	281958	1120	44656	58835	1836	369	44733	59328	950	10.04	42337	65536	631
case6	286234	391	57375	48739	993	364	46062	52845	836	11.11	39601	59247	745
case7	229149	406	37221	43490	577	377	26438	47981	515	9.67	25424	50109	354
case8	128544	388	19782	22846	504	383	17690	23564	286	11.81	15588	25826	467
case9	317581	1138	55399	66331	2045	383	56125	65417	1087	9.68	52304	68650	653
case10	102400	387	24381	18097	380	366	9990	19893	338	11.46	10153	22443	423
Average	-	788.5	44012.7	50899.5	1300.10	371.3	39948.9	**49957.2**	706.90	**11.18**	**38504**	53338	**558.7**
Ratio	-	1.000	1.000	1.000	1.000	0.471	0.911	**0.993**	0.544	**0.014**	**0.875**	1.048	**0.430**

† *We observed ±1% variation on Neural-ILT results (statistics of 50 rounds). The* `interpolate` *function in* `PyTorch` *involves* `atomicAdd` *which may be a source of non-determinism.*

[23] and compare the results with the conventional ILT [5] (denoted as ILT), and the SOTA learning-based OPC solution (denoted as PGAN-OPC [19]). Quantitative results are listed in Table 1. In Table 1, column "TAT" lists the turn around time for the **entire** end-to-end ILT optimization on a given input mask. Columns "L_2", "PVB", and "# shots" denote the squared L_2 error, the process variation band, and the mask fracturing shot count, respectively. Our Neural-ILT uses a consistent configuration for every test input, where the initial learning rate = 0.002, ILT iterations = 40, $\alpha = 1$ and $\beta = 0.3$. It can be observed that our Neural-ILT outperforms other approaches in terms of "TAT", "L_2" and "# shots" metrics. More specifically, comparing with ILT [5] and PGAN-OPC [19], Neural-ILT achieves 70×, 33× TAT speedup, 12.3%, 3.4% squared L_2 error reduction, and 67%, 21% mask fracturing shot count reduction, respectively. Figure 6 compares the complexity of masks synthesized by ILT, PGAN-OPC and Neural-ILT. As expected, Neural-ILT is able to generate masks with lower complexity comparing with ILT and PGAN-OPC. 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 comparable SOTA mask printability. It is worth mentioning that the significant speedup of Neural-ILT is mainly contributed from 3 aspects (in descending order of importance): (1) the acceleration by the CUDA-based litho-simulator; (2) the better initial solution predicted by the pre-trained UNet; and (3) higher searching efficiency of `PyTorch` built-in optimizer.

4.2 The Extensibility of Neural-ILT Paradigm

Our proposed ILT layers endow neural networks with the ability of self ILT correction. One should notice that the backbone network of Neural-ILT is not necessary an UNet. In order to demonstrate the extensibility/flexibility of the Neural-ILT paradigm, we further extend Neural-ILT to Neural-ILT-GAN, in which the pre-trained UNet is replaced by the generator of a pre-trained GAN (based on [27], one of the recent advances in GAN researches). Figure 7 depicts how do the L_2 loss curves of Neural-ILT and Neural-ILT-GAN change with respect to their on-neural-network ILT correction processes. Both Neural-ILT and Neural-ILT-GAN are tested using the same configuration, where the initial learning rate = 0.0005, ILT iterations = 80, $\alpha = 1$ and $\beta = 0.3$. Note that in Figure 7, Neural-ILT is able to obtain better initial solutions and better convergence comparing to Neural-ILT-GAN. Such phenomena may mainly due to our special training strategy of the pre-trained UNet (Section 3.5), which introduced domain knowledge as regularization. On the other hand, the pre-trained GAN is obtained following its original training recipe [27]. Nevertheless, the L_2 loss curves of both Neural-ILT and Neural-ILT-GAN still share the same decreasing trends with respect to the growth of the ILT iterations, which verified the feasibility of extending Neural-ILT paradigm to other network architectures. We believe such a flexibility can help designers introduce more domain-specific knowledge for achieving certain design objectives.

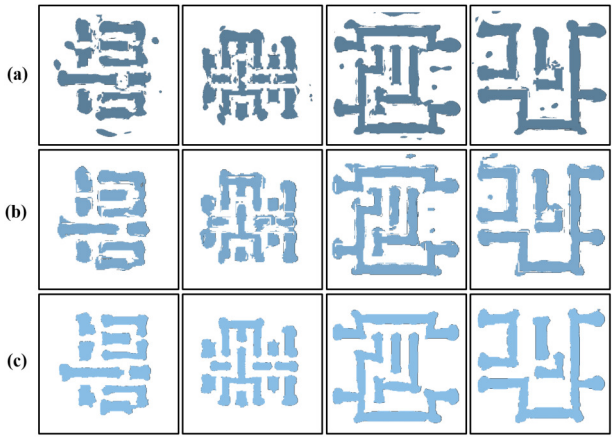


Figure 6: Mask complexity visualizations of (a) conventional ILT [5], (b) PGAN-OPC [19] and (c) Neural-ILT.

4.3 The Necessity of On-Neural-Network ILT

Here we want to have a short discussion on what really distinguishes Neural-ILT from general ILT. We designed an experiment for both conventional ILT and Neural-ILT on five additional benchmarks 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 [5] using our CUDA-based litho-simulation

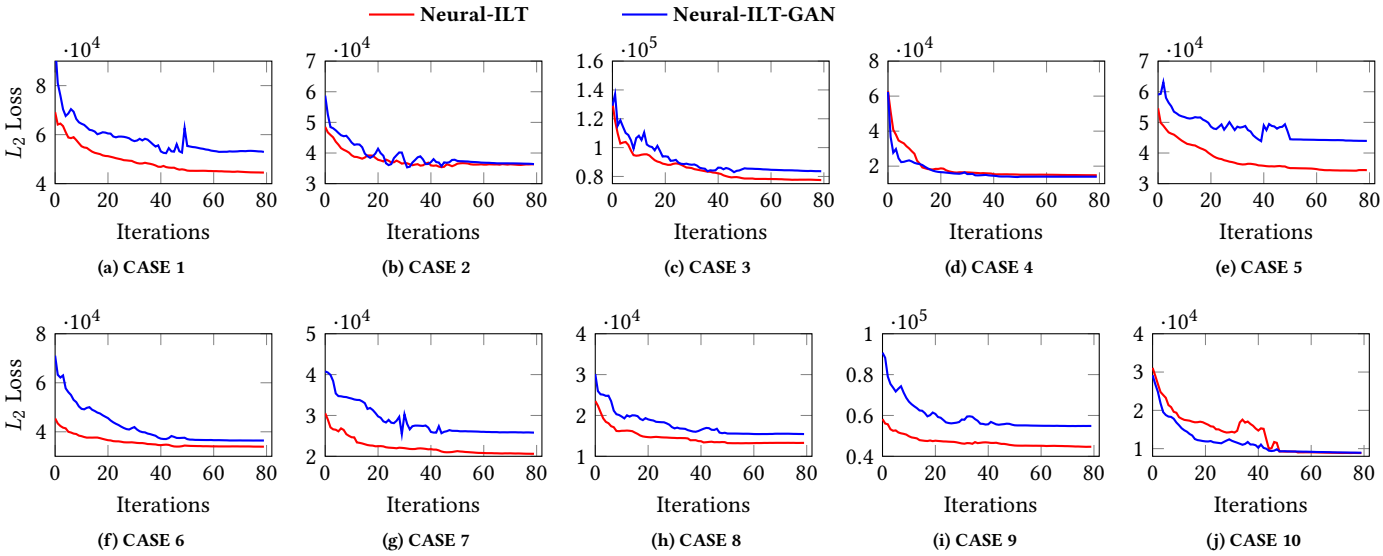


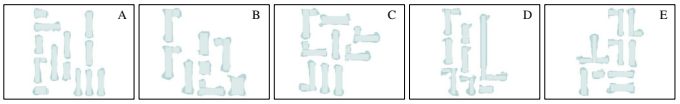
Figure 7: L_2 loss curves of Neural-ILT correction and Neural-ILT-GAN correction on ICCAD 2013 contest benchmarks.

Table 2: Mask Printability, Complexity and Runtime Performance Comparison between GPU-ILT and Neural-ILT

Bench	GPU-ILT			Neural-ILT		
ID	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
Average	18.81	111978.2	1258.2	11.99	101326.4	610
Ratio	1.00	1.00	1.00	**0.64**	**0.91**	**0.49**

* *Score* = L_2 (nm^2) + *PVBand* (nm^2), *which reflects the printability of a mask.*

(a) Direct fracturing results of GPU-ILT synthesized masks on A-E.



(b) Direct fracturing results of Neural-ILT synthesized masks on A-E.

Figure 8: Visualizations of mask fracturing results of (a) GPU-ILT, (b) Neural-ILT. Deeper color (equivalent to denser shots outlines) indicates more rectangular shots are required to accurately replicate the curvilinear ILT patterns.

tool. Thus, we can have a clearer insight on the intrinsic differences between on-neural-network ILT optimization (Neural-ILT) and on-mask-image ILT optimization (GPU-ILT). For each benchmark, Neural-ILT performs 40 iterations of corrections with learning rate = 0.001, while GPU-ILT performs 100 iterations of corrections with learning rate = 1. Quantitative results and corresponding visualizations are depicted in Table 1 and Figure 8. It's obvious that comparing to on-mask ILT (GPU-ILT), Neural-ILT consumes **less ILT iterations** (i.e., 100 vs. 40) with much **smaller learning rate** (i.e., 1.0 vs. 0.001), while achieving better overall quality (i.e, 9% better printability, 51% less mask shots counts). Recall that general ILT treats the mask optimization as an inverse problem of the lithography 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 (i.e., much smaller learning rate) on the weights and neurons' activities with considerably larger searching space. With the powerful capability of neural network in function approximation, the feed-forward 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 result in efficient computation. As the results, a converged Neural-ILT model, is cacheable, reusable and fine-tuneable, such features indicate that Neural-ILT has great potential of transferability among layouts with different data distributions, which can be further exploited in the future.

5 CONCLUSION

In this paper, we propose 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 data. Future works would include further studies on the necessity and applications of on-neural-network ILT.

REFERENCES

[1] H. Yang, W. Zhong, Y. Ma, H. Geng, R. Chen, W. Chen, and B. Yu, "VLSI mask optimization: From shallow to deep learning," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2020, pp. 434–439.

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

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

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

[5] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *ACM/IEEE Design Automation Conference (DAC)*, 2014, pp. 52:1–52:6.

[6] K. Hooker, B. Kuechler, A. Kazarian, G. Xiao, and K. Lucas, "ILT optimization of EUV masks for sub-7nm lithography," in *Proceedings of SPIE*, vol. 10446, 2017.

[7] Y. Ma, J.-R. Gao, J. Kuang, J. Miao, and B. Yu, "A unified framework for simultaneous layout decomposition and mask optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 81–88.

[8] R. Pearman, J. Ungar, N. Shirali, A. Shendre, M. Niewczas, L. Pang, and A. Fujimura, "How curvilinear mask patterning will enhance the EUV process window: a study using rigorous wafer+ mask dual simulation," in *Proceedings of SPIE*, vol. 11178, 2019.

[9] I. Torunoglu, A. Karakas, E. Elsen, C. Andrus, B. Bremen, B. Dimitrov, and J. Ungar, "A GPU-based full-chip inverse lithography solution for random patterns," in *Proceedings of SPIE*, 2010, p. 764115.

[10] V. Domnenko, B. Küchler, W. Hoppe, J. Preuninger, U. Klostermann, W. Demmerle, M. Bohn, D. Krüger, R. R. H. Kim, and L. E. Tan, "EUV computational lithography using accelerated topographic mask simulation," in *Proceedings of SPIE*, 2019.

[11] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[12] J. Kuang, J. Ye, and E. F. Y. Young, "Simultaneous template optimization and mask assignment for DSA with multiple patterning," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2016, pp. 75–82.

[13] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "LithoGAN: End-to-end lithography modeling with generative adversarial networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 107:1–107:6.

[14] Y. Lin, M. Li, Y. Watanabe, T. Kimura, T. Matsunawa, S. Nojima, and D. Z. Pan, "Data efficient lithography modeling with transfer learning and active data selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2019.

[15] D. Ding, B. Yu, J. Ghosh, and D. Z. Pan, "EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2012, pp. 263–270.

[16] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 6, pp. 1175–1187, 2019.

[17] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 146:1–146:6.

[18] 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," *Microelectronic Engineering*, vol. 168, pp. 15–26, 2017.

[19] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *ACM/IEEE Design Automation Conference (DAC)*, 2018, pp. 131:1–131:6.

[20] B. Jiang, H. Zhang, J. Yang, and E. F. Young, "A fast machine learning-based mask printability predictor for OPC acceleration," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019, pp. 412–419.

[21] T. Chan, P. Gupta, K. Han, A. A. Kagalwalla, and A. B. Kahng, "Benchmarking of mask fracturing heuristics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, pp. 170–183, 2017.

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

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

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. MICCAI*, 2015, pp. 234–241.

[25] W. Lv, S. Liu, Q. Xia, X. Wu, Y. Shen, and E. Y. Lam, "Level-set-based inverse lithography for mask synthesis using the conjugate gradient and an optimal time step," *JVSTB*, vol. 31, p. 041605, 2013.

[26] B. Jiang, X. Zhang, R. Chen, G. Chen, P. Tu, W. Li, E. F. Young, and B. Yu, "Fit: Fill insertion considering timing," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, p. 221.

[27] M.-Y. Liu, T. Breuel, and J. Kautz, "Unsupervised image-to-image translation networks," in *Advances in neural information processing systems*, 2017, pp. 700–708.