# DevelSet: Deep Neural Level Set for Instant Mask Optimization

Guojin Chen, Ziyang Yu, Hongduo Liu, Yuzhe Ma, *Member, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—As one of the key techniques for resolution enhancement technologies (RETs), optical proximity correction (OPC) suffers from prohibitive computational costs as feature sizes continue to shrink. Inverse lithography techniques (ILTs) treat the mask optimization process as an inverse imaging problem, yielding high-quality curvilinear masks. However, ILT methods often fall short of printability and manufacturability due to their time-consuming procedures and excessive computational overhead. In this article, we propose DevelSet, a potent metal layer OPC engine that replaces discrete pixel-based masks with implicit level set-based representations. With a GPU-accelerated lithography simulator, DevelSet achieves end-to-end mask optimization using a neural network to provide quasi-optimized level set initialization and further evolution with a CUDA-based mask optimizer for fast convergence. The backbone of DevelSet-Net is a transformer-based multibranch neural network that offers a parameter selector to eliminate the need for manual parameter initialization. Experimental results demonstrate that the DevelSet framework outperforms state-of-the-art approaches in terms of printability while achieving fast runtime performance (around 1 s). We expect this enhanced level set technique, coupled with a CUDA/DNN accelerated joint optimization paradigm, to have a substantial impact on industrial mask optimization solutions.**

***Index Terms*—Design for manufacture, level set, machine learning algorithms.**

## I. Introduction

OPTICAL proximity has gained prominence in the field of semiconductor lithography due to the observation of back-scattered electrons and proximity effects. The impact of the proximity effect becomes more pronounced as feature sizes decrease in advanced nodes, resulting in a decline in yield. To address this issue, a resolution enhancement technique known as optical proximity correction (OPC) has been developed. The technique helps to ensure pattern quality and improve printability on the wafer. The use of OPC has become increasingly important in modern semiconductor manufacturing, as it enables the production of smaller and more complex features with improved accuracy and yield. For years, photomask makers have used typical OPC approaches, including rule-based methods [1] and model-based methods [2], [3]. At advanced nodes, though, the OPC features are becoming smaller and more complex. More advanced resolution enhancement technologies (RETs) have been introduced, such as inverse lithography techniques (ILTs) [4], [5] and DNN-based methods [6], [7], [8]. Model-based OPC iteratively moves the fragmented mask segments assisted by the lithographic model to settle on the proper positions for the best convergence finally. Simple yet efficient, model-based methods dominate in early design automation software like Calibre [9]. ILTs can generate more robust, curvilinear support features on the mask. ILT also addresses mask optimization as an inverse imaging problem. It employs pixel-wise optimization to increase the process window's latitude and hence broaden the solution space. Gao et al. [4] derived a closed-form gradients descent technique using direct edge placement error (EPE) and optimization of the process window. Although Model-based and ILT-based methods have been widely adopted in industry, they inevitably suffer from heavy computational overhead because multiple rounds of lithography simulation are essential during the optimization process. This issue gets more acute with decreasing technology node and more complicated layout shapes.

Recently, emerging deep learning methods and models are widely applied on the EDA area. DNN-based methods are gradually becoming the mainstream in OPC research for their significant speedup and comparable mask printability. Choi et al. [10] constructed a classifier-based mask bias model to improve previous regression models. Yang et al. [6] proposed a generative model to produce an initial solution, which greatly lowers the number of iterations required in traditional ILT methods. LithoGAN [11] proposed an end-to-end lithography modeling approach using generative adversarial networks (GANs), which can accurately predict the output of the lithography process while considering nonlinearity and randomness factors. Chen et al. [7] first enabled full chip OPC by incorporating a DNN-based lithography simulator and mask generator simultaneously, which outperforms the industry toolkit on ISPD 2019 benchmark. A2-ILT [12] introduces a novel GPU-accelerated ILT algorithm that incorporates a spatial attention mechanism with reinforcement learning to improve accuracy and efficiency. Jiang et al. [13] presented an end-to-end mask optimization framework using self-training, which improves the accuracy of lithography simulation by iteratively refining the mask design. Recently, Yang et al. have proposed novel approaches to computational lithography that combine physics-based models with neural networks. Specifically, DOINN [14] presents a dual-band optics-inspired

Manuscript received 29 November 2022; revised 13 March 2023 and 26 May 2023; accepted 28 May 2023. Date of publication 14 June 2023; date of current version 22 November 2023. This work was supported by the Research Grants Council of Hong Kong, SAR, under Project CUHK14208021. The preliminary version has been presented at the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2021. This article was recommended by Associate Editor I. H.-R. Jiang. *(Corresponding author: Bei Yu.)*

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

Yuzhe Ma is with the Microelectronics Thrust, Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511453, China.

Digital Object Identifier 10.1109/TCAD.2023.3286262

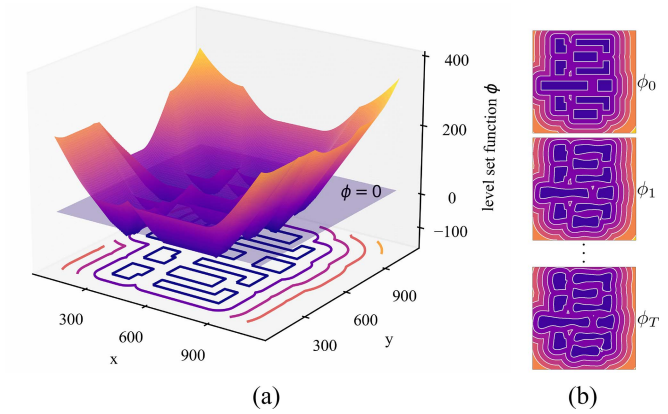


Fig. 1. (a) 3-D illustration of level set function $\phi$. The mask is shaped as the cross-section of the level set continuum with the zero plane. The contours on the $x$–$y$ plane are the projected level set. (b) Level set evolution process.

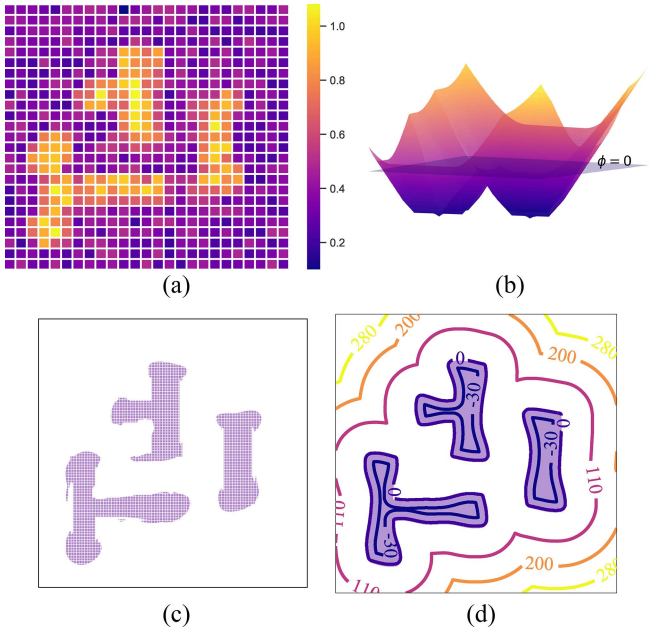


Fig. 2. Comparison of pixel-based ILT and level set-based ILT. (a) Intensity matrix of pixel-based ILT. (b) Level set-based ILT. (c) Mask generated by pixel-wise intensity threshold. (d) Mask generated by zero level set.

neural network that predicts lithography patterns for complex mask layouts with high accuracy and scalability, while [15] introduces a physics-inspired model that enables scalable computational lithography with high accuracy.

In the past decades, level set-based ILT methods have been actively explored as a feasible alternative to pixel-based ILT methods in OPC. As illustrated in Fig. 1, the level set continuum $\phi$ is a mathematical function used to represent a geometric object such as a surface or a curve in higher dimensions. Specifically in Fig. 1(a), the level set function associates the signed distance value with each pixel in the mask image, such that pixels with the same value lie on the same level set. The basic idea of level set-based ILT is to represent the solution of the mask optimization problem as the zero plane of a level set function, and then evolve this level set function over time to find the optimal solution [Fig. 1(b)]. The implicit representation of the level set method is naturally more effective in dealing with complex topology changes and lithography development [16]. Shen et al. [17] solved the inverse lithography problem using a level set time-dependent model with finite difference schemes. They further considered defocus and aberration to enhance robustness against process variations [18]. Lv et al. [19] improved the pattern fidelity with fast convergence by employing the conjugate gradient (CG) method and optimized time step. Geng et al. [20] adopted the process variation band (PVBand) cost function and reduced the runtime by leveraging the hybrid CG method. Yu et al. [21] proposed a momentum-based CG method and accelerated the level set evolution with a GPU-enabled fast Fourier transform (FFT) algorithm.

Briefly, we can abstract ILT approaches into two main categories: 1) parametric and 2) implicit. The parametric methods [4], [5], [6], [7], [8] use pixel-wise matrices to generate masks [Fig. 2(a)]. While the implicit approaches represent the mask as a zero level set cross-section [17], [18], [19], [20], [21] [Fig. 2(b)]. Due to the simplicity and adaptability of pixel-wise gradient descent methods, the parametric methods have been extensively studied from the viewpoints of the objective function, optimization method, and DNN acceleration, attaining state-of-the-art (SOTA) runtime performance and mask print fidelity. Nevertheless, as depicted in Fig. 2(c), parametric methods generate unnecessary isolated stains or edge glitches with zigzagging and tortuous complex mask boundaries, whereas the level set implicit representation is achieved with mask boundary continuity and curvature control [Fig. 2(d)]. However, the utilization of level set-based ILT approaches has been drastically underestimated owing to the added computational load brought about by level set evolution terms. Given the significant rise in processing power and CUDA-accelerated applications, it is time to investigate the vast potential of level set-based ILT techniques.

Motivated by these issues, we offer a dual acceleration architecture with GPU and neural networks based on level set mask optimization algorithms: deep neural level set (DevelSet), as shown in Fig. 6. There are two primary components of DevelSet: 1) DevelSet-Net (DSN) and 2) DevelSet-Optimizer (DSO). Inspiring by recent advancements in vision transformers (ViT) [22], [23], [24], [25], we use a transformer as the foundation for feature extraction. The input picture in ViT [22] is represented as a series of patches known as visual tokens. The visual tokens with positional embeddings are supplied into the transformer encoder network. Briefly, the proposed DSN leverages the transformer's local feature extraction capability to give quasi-optimal solutions using neural networks' quick inference capacity. In addition, DSN utilizes an extra unique modulation branch to permit DSO for curvature cost, as described in Section III-B1. To reduce mask complexity, DSO first inserts a curvature term into level set-based ILT. Furthermore, DSO employs a variety of GPU-friendly techniques to alleviate the computational strain during optimization iterations. Moreover, it is noticed that the printability performance is influenced by the various factors of the lithography process and the level set evolution process. In this article, we adopt *Transformer* as the parameter selector to enhance the whole DevelSet framework. Instead of depending on knowledgeable specialists to manually design the parameters, we propose a transformer-based parameter selector

TABLE I
SYMBOLS AND NOTATIONS USED THROUGHOUT THIS ARTICLE

| Symbols | Descriptions |
|---|---|
| $\mathbf{Z}_{\mathrm{t}}$ | Matrix representing the target layout pattern |
| $\mathbf{M}$ | Matrix representing the mask pattern |
| $\mathbf{I}$ | Matrix representing the aerial image |
| $\mathbf{Z}$ | Wafer image under nominal process condition |
| $\mathbf{Z}_{in}$ | Wafer image under min process condition |
| $\mathbf{Z}_{out}$ | Wafer image under max process condition |
| $\phi$ | Level set function (LSF) |
| $\mathbf{H}$ | Lithography kernels |

that can automatically prepare parameters for DSO development to achieve optimal mask printability. The DevelSet architecture is enhanced by the end-to-end joint optimization of DSN and DSO, enabling SOTA fast convergence and mask printability.

Our main contributions are as follows.

1) We propose DevelSet, an improved level set-based ILT framework with CUDA and DNN acceleration.
2) We first introduce curvature term into level set-based ILT methods to reduce mask complexity and leverage GPU to perform all the calculations.
3) To the best of our knowledge, it is the first time to integrate level set into deep neural network for an end-to-end joint mask optimization flow.
4) We design a novel transformer-based multibranch neural network architecture with level set embeddings to further boost the performance and improve mask printability.
5) We leverage the transformer model for the parameter selection task to enhance the performance of the DevelSet framework.
6) Experimental results show that DevelSet achieves SOTA mask printability with predominant runtime advantage for instant mask optimization.

The remainder of this article is organized as follows. Section II lists some preliminaries about level set algorithms and mask optimization methods. Section III details the DevelSet algorithm. Section IV presents our experimental results, followed by a conclusion in Section V.

## II. PRELIMINARIES

In this section, we will introduce concepts and background related to this work. Following the traditions, major math symbols and their descriptions are listed in Table I.

### A. Level Set-Based ILT Algorithms

In recent years, level set methods [26] have been actively researched as a feasible alternative for pixel-based ILT methods. When applying the level set methods for mask optimization in 2-D space $\Omega$, let $\mathcal{C}: \Omega \to \mathbb{R}^2$ denote a parametric curve in 2-D space $\Omega$, we can denote the boundary of the input mask using an implicit function $\phi(x, y): \Omega \to \mathbb{R}$

$$\mathcal{C} = \{(x, y) \mid \phi(x, y) = 0\} \tag{1}$$

where $\phi(x, y)$ is called the level set function. The level set evolution process for mask optimization is illustrated in Fig. 1(b), which depicts the bird's-eye view of the crossing layer between the zero plane and the level set function $\phi(x, y)$. Mathematically, the level set-based mask optimization can be derived as the evolution along the descent of the LSF $\phi$. The commonly used LSF $\phi$ is the signed distance function (SDF)

$$\phi_{\mathrm{SDF}}(x, y) = \begin{cases} -d(x, y), & \text{if } (x, y) \in \text{inside}(\mathcal{C}) \\ 0, & \text{if } (x, y) \in \mathcal{C} \\ d(x, y), & \text{if } (x, y) \in \text{outside}(\mathcal{C}) \end{cases} \tag{2}$$

where $d(x, y)$ is the minimum Euclidean distance from point $(x, y)$ to the parametric curve $\mathcal{C}$. As illustrated in Fig. 2(d), the contours are labeled with its SDF values, and the $\mathcal{C}$ is the contour labeled by 0. Now, the mask image $\mathbf{M}$ can be represented by $\phi$ as

$$\mathbf{M}(x, y) = \begin{cases} 1, & \text{if } \phi(x, y) \le 0 \\ 0, & \text{if } \phi(x, y) > 0. \end{cases} \tag{3}$$

Let $\mathcal{C}(t)$ denote a curve that depends on a time parameter $t \in \mathbb{R}$. Specifically, in mask optimization, $\mathcal{C}$ is mask boundary. $\mathcal{C}(t)$ represents different mask boundaries for $t \in \{0, 1, 2, \ldots, T-1\}$ iterations. Here, $T$ represents the total number of evolution steps. The curve evolution then can be formally defined as

$$\frac{\partial \mathcal{C}(t)}{\partial t} = v\boldsymbol{n} \tag{4}$$

where $\boldsymbol{n} = (\nabla\phi/|\nabla\phi|)$ is the unit vector in the outward normal direction of the curve $\mathcal{C}$ and $v$ indicates the velocity along the normal direction. We use the zero level set to implicitly represent the mask boundary, thus: $\phi(\mathcal{C}(t), t) = 0$. The chain rule gives us

$$\frac{\partial \phi(\mathcal{C}(t), t)}{\partial t} = 0 \to \frac{\partial \phi}{\partial \mathcal{C}(t)} \frac{\partial \mathcal{C}(t)}{\partial t} + \frac{\partial \phi}{\partial t} = 0. \tag{5}$$

Consider all the points on the evolving front $\mathcal{C}(t)$, $(\partial\phi/\partial\mathcal{C}) = \nabla\phi$, combining (4) and (5), the motion equation of LSF $(\partial\phi/\partial t)$ can be formally expressed by

$$\frac{\partial \phi}{\partial t} = -v|\nabla\phi|. \tag{6}$$

Equation (6) is a partial differential equation (PDE). Once the level set $\phi$ and velocity $v$ are defined, the first-order derivative in space and time of (6) can be approximated using finite difference techniques. Evolution of LSF $\phi(x, y, t)$ can be performed iteratively. We use $\phi_i(x, y)$ to denote $\phi(x, y, t_i)$ for simplicity. For $i \in \{0, 1, 2, \ldots, T-1\}$, the $i$th-step update is

$$\phi_{i+1}(x, y) = \phi_i(x, y) + \Delta t \frac{\partial \phi_i}{\partial t} \tag{7}$$

where $\Delta t$ is the time step, $\phi_0(x, y)$ is the initial LSF, and the $\phi_T(x, y)$ is the corresponding output LSF after $T$ evolution steps. As shown in Fig. 1(b), we can obtain the optimized mask after $T$ steps by applying (3).

### B. Lithography Simulation Model

Lithography simulation techniques are crucial to modern IC manufacturing, which consists of the optical simulation and resist simulation. In the conventional optical simulation process, an aerial image can be calculated through a mask pattern $\mathbf{M}$ using an optical projection printing system. The

optical model is characterized by the illumination type and projection lenses of an exposure tool, which can be expressed mathematically using Hopskins' diffraction model [27]. The sum of coherent systems (SOCSs) can roughly estimate Hopskins' diffraction model by performing singular value decomposition. The optical projection process is then replaced by a set of coherent kernels. The intensity of aerial image $\mathbf{I}$ can be represented by convolving the mask $\mathbf{M}$ and a set of optical kernels $\mathbf{H}$

$$\mathbf{I}(x, y) = \sum_{i=1}^{N^2} \sigma_i |\mathbf{M}(x, y) \otimes h_i(x, y)|^2. \tag{8}$$

Here, $\otimes$ denotes the convolution operation. $h_i$ is the $i$th kernel of the optical kernel set $\mathbf{H}$ $\sigma_i$ is the corresponding weight of the coherent system, and $(x, y)$ is index notation of matrix. $M(x, y)$ is the pixel value at the point $(x, y)$ of mask image $\mathbf{M}$. The $N_k$th-order approximation to the partially coherent system can be obtained by

$$\mathbf{I}(x, y) \approx \sum_{i=1}^{N_k} \sigma_i |\mathbf{M}(x, y) \otimes h_i(x, y)|^2 \tag{9}$$

where $N_k = 24$ in our implementation. After optical simulation, the aerial image undergoes a resist model to estimate the final printed shape on the wafer. In the resist stage, the distribution of 2-D light intensity $\mathbf{I}$ incident on top of the photoresist material on the wafer plane. For methodology verification and also for simplicity, we adopt the constant threshold resist model which is consistent with the ICCAD 2013 contest settings [28]. As depicted in Fig. 2(c), given the print threshold $I_{\text{th}}$, the printed wafer image can be expressed as

$$\mathbf{Z} = \begin{cases} 1, \text{ if } \mathbf{I} \geq I_{\text{th}} \\ 0, \text{ if } \mathbf{I} < I_{\text{th}}. \end{cases} \tag{10}$$

### C. Mask Printability and Mask Manufacturability

In this article, we use squared $L_2$ error and PVBand as two typical metrics to evaluate mask printability. Mask printability represents the quality of the printed patterns generated from the optimized mask. Moreover, the mask fracturing shot count proposed in Neural-ILT [8] is also applied in this work to evaluate mask complexity and manufacturability.

*1) Squared $L_2$ Error:* Given the wafer image $\mathbf{Z}$ and target image $\mathbf{Z}_t$, the squared $L_2$ error is calculated by

$$L_2(\mathbf{Z}, \mathbf{Z}_t) = \|\mathbf{Z} - \mathbf{Z}_t\|_2^2. \tag{11}$$

*2) PVBand:* PVBand is the bitwise-XOR region among all the printed patterns under different process conditions. In our work, for simplicity, we calculate the PVBand under two extreme conditions, one at nominal condition with $+2\%$ dose and the other one at defocus and $-2\%$ dose. A mask is more robust if its PVBand area is smaller.

*3) EPE:* EPE refers to the deviation between the intended and actual positions of features on a chip, resulting from variations in the lithography, etching, and deposition processes. In accordance with [28], the calculation of EPE involves sampling a series of measurement points along the contour of the target design. Illustrated in Fig. 3 and expressed in (12), if the distance $D(x, y)$ between the target design and the printed image exceeds a predefined EPE threshold, denoted as *thres_epe*, the measurement point $(x, y)$ is classified as an EPE violation. Following [28], the value of *thres_epe* is set to 15 nm (equivalent to 15 pixels)

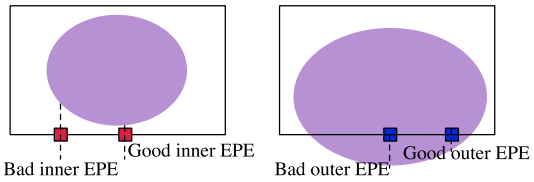


Fig. 3. Visualization of EPE errors.

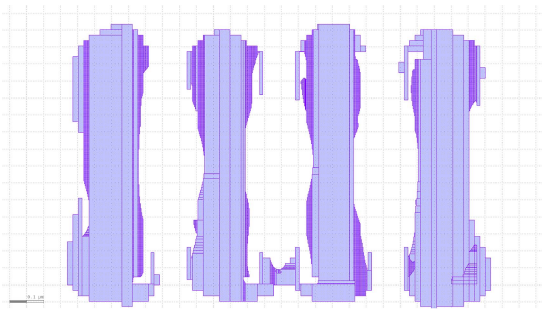

Fig. 4. Mask fracturing shot count. Split the mask $\mathbf{M}$ into a collection of small rectangles that can completely cover the original mask.

$$\text{EPE}(x, y) = \begin{cases} 1, D(x, y) \geq \textit{thres\_epe} \\ 0, D(x, y) < \textit{thres\_epe}. \end{cases} \tag{12}$$

*4) Mask Fracturing Shot Count:* Numerous classic pixel-based ILT techniques have a tendency to optimize the mask solely for printability. However, the majority of these optimized masks contain an abundance of minute irregular sub-features, which makes mask fabrication more difficult. In this work, the shot count is used to assess the mask's manufacturability. An evaluated mask $\mathbf{M}$ can be fragmented into a collection of little rectangles that replicate the original mask exactly, as depicted in Fig. 4. The number of mask fracturing shots represents the number of broken rectangles.

### D. Transformer and Attention Mechanism

As a competitive substitute for convolutional neural networks (CNNs), which are currently SOTA in computer vision and are consequently widely employed in various image identification applications, the ViT [22] models outperform the current SOTA CNN by almost $\times 3$ in terms of performance and efficiency. ViT is based on transformer architecture which has become a de-facto standard in neural language processing (NLP). CNNs process images as arrays of pixels, but ViT employs fixed-size image patches to allow sequence modeling and assist parallel processing. The positional encoded image patches, also known as visual tokens, will serve as the input for the transformer encoder. The exceptional performance of the transformer encoder depends on the self-attention mechanism, which enables the encoding of global information while maintaining a focus on the most relevant local features.

*1) Attention Mechanism:* Self-attention is one of the fundamental building components of machine learning transformers. It is a mathematical primitive designed to automatically measure paired entity interactions, which aids a network in

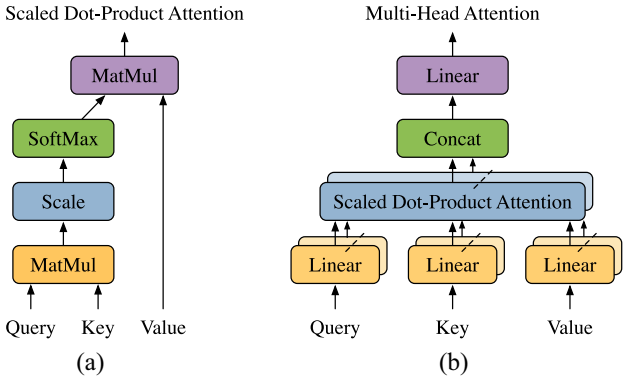


Fig. 5. Two basic modules for transformer architecture. (a) Scaled dot-product attention. (b) Multihead attention.

discovering hierarchies and alignments among input data. As illustrated in Fig. 5, the attention layer takes its input in the form of three parameters, known as the Query, Key, and Value. For the multihead attention [Fig. 5(b)], the module splits its Query, Key and Value ($\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d_m}$) inputs N-ways and pass each split into a core module, which is called Scaled Dot-Product Attention in Fig. 5(a). We formulate the multihead attention as

$$\mathrm{MHA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Concat}(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_h)\boldsymbol{W} \tag{13}$$

where $n$ is the length of sequence and $d_m$ is the dimension for each element of the sequence. $\boldsymbol{H}_i, i \in \{1, 2, \ldots, h\}$ is the output of the single scaled dot-product attention head, which is formulated as follows:

$$\begin{aligned} \boldsymbol{H}_i = & \ \mathrm{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right) \\ = & \ \mathrm{softmax}\left[\frac{\boldsymbol{Q}\boldsymbol{W}_i^Q(\boldsymbol{K}\boldsymbol{W}_i^K)^\top}{\sqrt{d_k}}\right]\boldsymbol{V}\boldsymbol{W}_i^V \end{aligned} \tag{14}$$

where $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K \in \mathbb{R}^{d_m \times d_k}, \boldsymbol{W}_i^V \in \mathbb{R}^{d_m \times d_v}$. And the weight matrix in (13) $\boldsymbol{W} \in \mathbb{R}^{hd_v \times d_m}$.

## III. Deep Neural Level Set Algorithms

The whole architecture is depicted in Fig. 6, which could be separated into three sections and includes the CUDA accelerated truncated SDF (TSDF), DSO, and DSN components. Using the CUDA accelerated TSDF function, the input mask will initially be converted to a level set function. Then, depending on the input TSDF, DSN will forecast the initial solution, modulation branch, and optimal parameter configuration. All DSN outputs will be input into DSO for iterative evolutions in order to produce the final masks. This section will be structured as follows: Section III-A will begin with a detailed introduction for DSO, including the enhanced level set-based ILT algorithm, the curvature term to improve the mask manufacturability, and the complete implementation in the CUDA platform. We develop a set of efficient, GPU-friendly algorithms by combining the GPU parallelism mechanism with the numerical level set setup. Next, Section III-B proposes a novel transformer-based multibranch network, i.e., DSN. Each branch is meticulously crafted based on the characteristics of the level set-based ILT algorithm. The level set branch intends to provide a more efficient initial level set function for DSO convergence. The modulation branch predicts a weighted attention matrix to selectively regularize the mask border, compensating for the mask printability loss resulting from the curvature term. Another crucial domain-specific parameter selector is introduced in Section III-C to generate the best parameter set for DSO to get the most optimized output mask. Finally, we perform the end-to-end joint optimization for DevelSet in Section III-D to accomplish instant mask optimization with higher mask printability and lower mask complexity.

### A. DevelSet-Optimizer

Conventional pixel-based ILT methods can be formulated as $\mathbf{M}_{\mathrm{opt}} = \mathcal{L}^{-1}(\mathbf{Z}_t; \mathbf{P}_{\mathrm{nom}})$, where $\mathbf{M}_{\mathrm{opt}}$ is optimized mask, $\mathcal{L}(\cdot; \mathbf{P}_{\mathrm{nom}})$ denotes the forward lithography process under the nominal condition, $\mathbf{Z}_t$ is the design target. After determining the objective errors, the pixel-based ILT approaches will update the intensity matrix pixel-by-pixel using gradient back-propagation from the lithography model. Pixel-based ILT algorithms are computationally costly and difficult to regulate intricacies of masks because to these qualities. Previous SOTA pixel-based approach Neural-ILT [8] built a CUDA accelerated lithography simulator and incorporated the GPU-accelerated litho-simulation module into neural networks to accomplish an on-neural-network ILT training solution. To lower the complexity of the mask, an ILT correction layer and a mask complexity refinement layer are applied. Nevertheless, compared to earlier learning-based work PGAN-OPC [6], Neural-ILT compromises printability for the sum of $L_2$ and PVBand is somewhat inferior.

In contrast to pixel-based ILT approaches, the evolution of the level set continuum [Fig. 2(b)] can be viewed as the mask optimization procedure. The mask is formed by the intersection of the zero plane with the level set continuum [Fig. 2(d)]. Mathematically, the level set continuum is denoted by LSF $\phi$, and the evolution technique is denoted by (7). Prior level set-based SOTA approach GLS-ILT [21] incorporated momentum term to regularize the level set optimization procedure. In addition, GPU was used to accelerate a portion of the lithography model. However, the acceleration impact was not noticeable due to the fact that only the FFT module was accelerated. Since data transfer between CPU and GPU consumes the majority of total runtime, there is still ample space for improvement.

DSO is a fully GPU-accelerated iterative mask optimizer that addresses the shortcomings of the prior technique. The improved level set ILT technique with TSDF regulates the mask's complexity and the mask's bounds with a curvature term. All computations are shifted to the GPU for accelerated performance. As a result, DSO simultaneously delivers better mask printability and runtime performance. According to (7), the level set function $\phi$ and the velocity term $v$ are the two essential components of level set evolution. As for the LSF $\phi$ in DSO, we employ the TSDF rather than the frequently applied SDF. In addition, we integrate the ILT-based gradient into the velocity term $v$ and introduce the curvature term to enhance the printability of the mask and minimize its complexity.

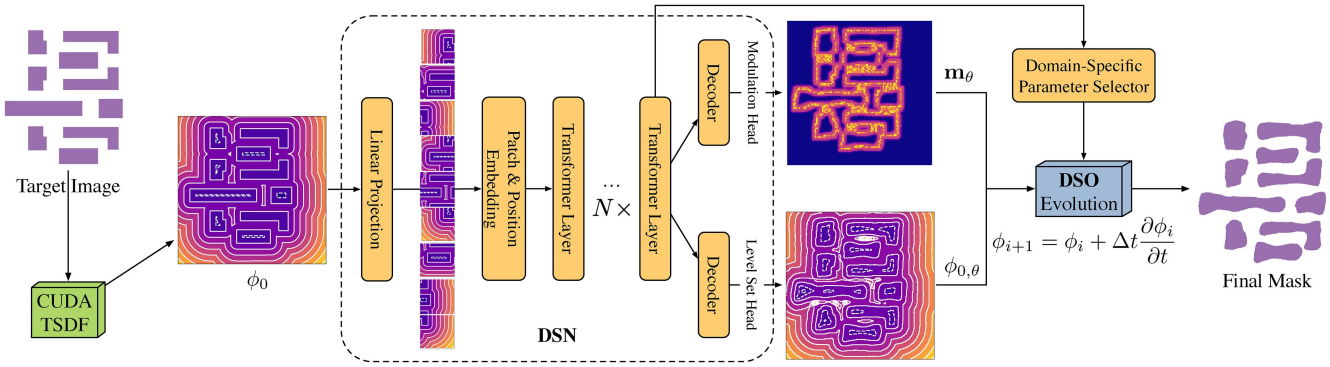


Fig. 6. Overview of DevelSet framework with the end-to-end joint optimization flow of DSN and DSO.

**Algorithm 1** DSO

**Require:** Target image $\mathbf{Z}^*$, optical kernels $h$;
**Ensure:** Optimized mask $\mathbf{M}^*$;
1: $\mathbf{M}_0 \leftarrow \mathbf{Z}^*$;
2: $\phi_0 \leftarrow \mathbf{M}_0$;
3: $\mathbf{G}_0 \leftarrow \frac{\partial \mathbf{G}_0(M)}{\partial \mathbf{M}_0}$;
4: $\mathbf{v}_0 \leftarrow -\mathbf{G}_0|\nabla\phi|$;
5: **repeat**
6: **for** all sites in 2-D pattern, parallel computing: **do**
7: **forward:**
8: Curvature: $\kappa_i \leftarrow \nabla \frac{\nabla\phi_i}{|\nabla\phi_i|}$;
9: Change: $\Delta\phi_i \leftarrow (\mathbf{v}_i + \kappa_i)\delta t$;
10: Level set function: $\phi_{i+1} \leftarrow \phi_i + \Delta\phi_i$;
11: Mask pattern: $\mathbf{M}_{i+1} \leftarrow \phi_{i+1}$;
12: Resist pattern: $\mathbf{Z}_{i+1} \leftarrow \mathbf{M}_{i+1}$;
13: **backward:**
14: Gradient of cost function: $\mathbf{G}_{i+1} \leftarrow \frac{\partial \mathbf{G}_{i+1}(\mathbf{M})}{\partial \mathbf{M}}$;
15: Update: $\mathbf{v}_{i+1} = -\mathbf{G_{i+1}}(\mathbf{M})|\nabla\phi_{i+1}| + \lambda \cdot \mathbf{v}_i$;
16: **end for**
17: **until** $|\mathbf{v}|_{\max} < \epsilon$

Algorithm 1 demonstrates the forward and backward process of DSO. The TSDF $\phi$ in line 2, motion term $\mathbf{v}$ in line 4, and curvature term $\kappa$ are introduced in Section III-A1. Implementation details are addressed in Section III-A2 and Algorithm 2.

*1) Improved Level Set-Based ILT (Truncated Signed Distance Function):* Theoretically, as demonstrated in Section II-A, the level set framework is independent of the particular LSF $\phi$ chosen. SDF in (2) is a popular form of LSF since it is Lipschitz-continuous and practically everywhere differentiable. Nonetheless, when the level set evolves, the absolute value of the extremum of SDF may become excessively big, which is ineffective because we only need the level set with zero height to build the mask. Even worse, excessive extreme values can destabilize the training procedure, preventing convergence. Therefore, we adopt the TSDF as our LSF with upper and lower bounds $D_u$ and $D_l$, respectively

$$\phi_{\text{TSDF}} = \begin{cases} D_u, & \text{if } \phi_{\text{SDF}} > D_u \\ \phi_{\text{SDF}}, & \text{if } D_l \le \phi_{\text{SDF}} \le D_u \\ D_l, & \text{if } \phi_{\text{SDF}} < D_l. \end{cases} \tag{15}$$

**Algorithm 2** CUDA Level Set Algorithms

**Require:** Target image $\mathbf{Z}_\text{t}$
**function** CUDA_TSDF($\mathbf{Z}_\text{t}$)
$\mathbf{Z}_{tu}, \mathbf{Z}_{td} \leftarrow$ Shift $\mathbf{Z}_\text{t}$ upwards, downwards by 1 pixel;
$\mathbf{Z}_{tl}, \mathbf{Z}_{tr} \leftarrow$ Shift $\mathbf{Z}_\text{t}$ leftwards, rightwards by 1 pixel;
$b_h \leftarrow (\mathbf{Z}_\text{t}$ XOR $\mathbf{Z}_{tu}) + (\mathbf{Z}_\text{t}$ XOR $\mathbf{Z}_{td})$;
$b_v \leftarrow (\mathbf{Z}_\text{t}$ XOR $\mathbf{Z}_{tl}) + (\mathbf{Z}_\text{t}$ XOR $\mathbf{Z}_{tr})$;
**for all** pixels on target image $\mathbf{Z}_\text{t}$ **do**
$d_{ij} \leftarrow$ Distance from pixel $p_i$ to boundary $b_j$;
$d_i \leftarrow$ Minimum distance of point $p_i$ in all $d_{ij}$;
$\phi_{\text{SDF}} \leftarrow$ SDF matrix from all $d_i$;
$\phi_{\text{TSDF}} \leftarrow$ TSDF matrix using Equation (15);
**end for**
**return** $\phi_{\text{TSDF}}$;
**end function**
**Ensure:** Truncated Signed Distance Function $\phi_{\text{TSDF}}$;

**Require:** TSDF matrix $\phi_{\text{TSDF}}$;
**function** CUDA_geometry_gradient($\phi$)
$\phi_u, \phi_d \leftarrow$ Shift $\phi$ upwards, downwards by 1 pixel;
$\phi_l, \phi_r \leftarrow$ Shift $\phi$ leftwards, rightwards by 1 pixel;
$\nabla\phi_x \leftarrow (\phi_r - \phi_l)/2$; $\nabla\phi_y \leftarrow (\phi_u - \phi_d)/2$;
**return** $\nabla\phi_x, \nabla\phi_y$;
**end function**
**Ensure:** Geometry gradient $\nabla\phi_x, \nabla\phi_y$;

**Require:** TSDF $\phi_{\text{TSDF}}$, geometry gradient $\nabla\phi_x, \nabla\phi_y$;
**function** CUDA_curvature($\phi, \nabla\phi_x, \nabla\phi_y$)
$\nabla\phi_{xx} \leftarrow$ CUDA_geometry_gradient($\nabla\phi_x$);
$\nabla\phi_{yy} \leftarrow$ CUDA_geometry_gradient($\nabla\phi_y$);
$\phi_{ul}, \phi_{ur}, \phi_{dl}, \phi_{dr} \leftarrow$ Shift $\phi$ to 4 diagonal directions;
$\nabla\phi_{xy} \leftarrow ((\phi_{ur} - \phi_{ul}) - (\phi_{dr} - \phi_{dl}))/4$;
$\kappa \leftarrow$ Curvature term using Equation (27);
**return** $\kappa$;
**end function**
**Ensure:** Curvature term $\kappa$;

In our work, $D_u$ is set to 900 and $D_l$ is $-100$, according to the design rules of the benchmark. The TSDF increases the optimization procedure's stability by lowering the dataset's variance. In addition, the TSDF allows rapid convergence of the DSN to enable end-to-end joint optimization of the entire mask optimization framework.

*Motion Term:* Another core component in level set methods is the motion term $(\partial\phi/\partial t)$, which is mainly related to the velocity term $v$, according to (6). We move the level set continuum using the backpropagated gradient from the lithography simulator of the partial coherent imaging system. And the objective function of DSO incorporates ILT correction loss and PVBand loss

$$L_{\text{DSO}} = \alpha L_{\text{ilt}} + \beta L_{\text{pvb}}. \tag{16}$$

The ILT correction loss is intended to minimize the pixel-based difference between the output nominal image and the input target, which can be calculated as follows:

$$L_{\text{ilt}} = \sum_{x=1}^{N}\sum_{y=1}^{N}(\mathbf{Z}(x,y) - \mathbf{Z}_t(x,y))^2 \tag{17}$$

where $\mathbf{Z}_t$ is the target image; $\mathbf{Z}$ is the wafer image after the lithography under the nominal condition; $N$ is the width of the target image. To enable the evolution process differentiable, the step function in (10) is approximated as

$$\mathbf{Z} = \frac{1}{1+\exp(-\sigma_z \times (\mathbf{I} - I_{th}))} \tag{18}$$

where $\sigma_z$ is the steepness of the sigmoid function. Then, the gradient of ILT loss can be expressed as

$$\begin{aligned}\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}} &= 2\times(\mathbf{Z}-\mathbf{Z}_t)\odot\frac{\partial \mathbf{Z}}{\partial \mathbf{M}}\\ &= 2\sigma_z \times \left\{\mathbf{H}'\otimes\left[(\mathbf{Z}-\mathbf{Z}_t)\odot\mathbf{Z}\odot(1-\mathbf{Z})\odot\left(\mathbf{M}\otimes\mathbf{H}^*\right)\right]\right.\\ &\quad \left.+\left(\mathbf{H}'\right)^*\otimes\left[(\mathbf{Z}-\mathbf{Z}_t)\odot\mathbf{Z}\odot(1-\mathbf{Z})\odot(\mathbf{M}\otimes\mathbf{H})\right]\right\}\end{aligned} \tag{19}$$

where $\mathbf{H}'$ is the flipped optical kernel set $\mathbf{H}$, and $\mathbf{H}^*$ is the conjugate of $\mathbf{H}$.

To minimize the area of PVBand, we expect the innermost/outermost wafer under min/max process conditions as close to the target image as possible. The PVBand loss is given by

$$L_{\text{pvb}} = (\mathbf{Z}_{\text{in}} - \mathbf{Z}_t)^2 + (\mathbf{Z}_{\text{out}} - \mathbf{Z}_t)^2. \tag{20}$$

The gradient of PVBand loss can be represented as

$$\begin{aligned}\frac{\partial L_{\text{pvb}}}{\partial \mathbf{M}} &= 2\times(\mathbf{Z}_{\text{in}}-\mathbf{Z}_t)\odot\frac{\partial \mathbf{Z}_{\text{in}}}{\partial \mathbf{M}}\\ &\quad + 2\times(\mathbf{Z}_{\text{out}}-\mathbf{Z}_t)\odot\frac{\partial \mathbf{Z}_{\text{out}}}{\partial \mathbf{M}}.\end{aligned} \tag{21}$$

The detailed derivation of (21) is similar to (19). Now, the velocity $v$ is

$$v = \alpha\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}} + \beta\frac{\partial L_{\text{pvb}}}{\partial \mathbf{M}}. \tag{22}$$

And the motion equation is finally derived as

$$\frac{\partial \phi_i}{\partial t} = -\left(\alpha\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}} + \beta\frac{\partial L_{\text{pvb}}}{\partial \mathbf{M}}\right)|\nabla\phi_i|. \tag{23}$$

*Curvature Term:* As described in (7), the evolution method of a level set is specified by a number of updating terms that can be loosely classified into two groups: 1) external terms and 2) internal terms. The inverse lithography gradient or optimization methods are external terms that relocate the boundaries to the optimal place based on the loss function or data evidence. The internal terms emphasize the regularization of the curve shape, e.g., the curvature term or the curvature length. All prior level set-based algorithms address the improvement of well-researched external terms. The research on internal terms, which necessitates calculations of second-order derivatives, faces the challenge of a massive computing volume. In this article, DSO makes full advantage of the effective feature in the implicit representation to derive the curvature of the borders, which is useful for controlling the smoothness of the front and removing noisy points from the mask pattern. The curvature term is formally defined as

$$\kappa = \lambda\mathbf{m}_\theta|\nabla\phi_i|\,\text{div}\left(\frac{\nabla\phi_i}{|\nabla\phi_i|}\right) \tag{24}$$

where $\lambda$ is the weight of curvature. However, applying the curvature term straight to all situations will hurt the mask optimization efficiency, as masks should contain some sharp edges. In the majority of cases, high curvature should be penalized to decrease mask complexity. To regulate the curvature term, we add a weighted matrix $\mathbf{m}_\theta$, where subscript $\theta$ signifies that $\mathbf{m}_\theta$ is predicted by the modulation branch parameters of DSN. The modulation branch of Section III-B1 will be introduced in detail. The $\mathbf{m}_\theta$ has the same dimensions as the mask image $\mathbf{M}$ that the modulation branch of DSN would forecast. The level set evolution of DSO can then be expressed as the sum of the motion term and the curvature term

$$\begin{aligned}\frac{\partial \phi_i}{\partial t} &= -\left(\alpha\frac{\partial L_{\text{ilt}}}{\partial \mathbf{M}} + \beta\frac{\partial L_{\text{pvb}}}{\partial \mathbf{M}}\right)|\nabla\phi_i|\\ &\quad + \lambda\mathbf{m}_\theta|\nabla\phi_i|\,\text{div}\left(\frac{\nabla\phi_i}{|\nabla\phi_i|}\right).\end{aligned} \tag{25}$$

*2) Implementation of DSO Based on CUDA:* Conventional ILT-based mask optimization techniques incur a substantial computational expense, which is exacerbated by level set-based techniques. When new terms are utilized to improve mask printability, the already burdensome computing system incurs additional computational costs. Consequently, the primary obstacle for DSO is to deal with expensive computations. The rapid growth of general-purpose graphic processing unit (GPGPU) computing technology has resulted in the continual enhancement of GPU parallel processing capability. By putting the entire DSO architecture on the CUDA platform, we are able to achieve a balance between performance and efficiency. In this section, we will describe the CUDA implementation of each term in the level set method, as well as the engineering features that make our DSO framework considerably quicker.

*Numerical Settings:* The level set-based mask optimization methods focus on 2-D situation with an image as the input. The space is discretized by a Cartesian grid with steps $\Delta x$, $\Delta y$, where the coordinates $(x, y)$ represent the $x$th, $y$th pixel in the image. The first-order derivatives in space and time of (25) can be approximated using finite difference techniques. We apply weighted essential nonoscillatory (WENO) [29] numerical polynomial interpolation method that uses the smoothest possible polynomial interpolation to find $\phi$. And the first-order and second-order spatial derivatives of $\phi$ can be represented with central differences as

$$\nabla\phi_x = \frac{1}{2}(\phi(x+1,\ y) - \phi(x-1,\ y))$$

$$\nabla\phi_y = \frac{1}{2}(\phi(x,\ y+1) - \phi(x,\ y-1))$$

$$\begin{aligned}\nabla\phi_{xx} &= \phi(x+1,\ y) + \phi(x-1,\ y) - 2\times\phi(x,\ y)\\ \nabla\phi_{yy} &= \phi(x,\ y+1) + \phi(x,\ y-1) - 2\times\phi(x,\ y)\\ \nabla\phi_{xy} &= \frac{1}{4}\left[(\phi(x+1,\ y+1) - \phi(x-1,\ y+1))\right.\\ &\quad \left. - (\phi(x+1,\ y-1) - \phi(x-1,\ y-1))\right] \quad (26)\end{aligned}$$

and the curvature term is then computed numerically with

$$\begin{aligned}\kappa &= \lambda\mathbf{m}_\theta|\nabla\phi_i|\,\mathrm{div}\left(\frac{\nabla\phi_i}{|\nabla\phi_i|}\right)\\ &= \lambda\mathbf{m}_\theta\frac{\nabla\phi_{xx}\nabla\phi_y{}^2 - 2\nabla\phi_y\nabla\phi_x\nabla\phi_{xy} + \nabla\phi_{yy}\nabla\phi_x{}^2}{\nabla\phi_x{}^2 + \nabla\phi_y{}^2}. \quad (27)\end{aligned}$$

*CUDA-Based TSDF:* The first formidable obstacle is efficiently calculating the TSDF on a 2048-by-2048-pixel target image. Introduced by [30], the Fast Marching Method is the most renowned method for calculating SDFs. Instead of employing the Fast Marching Method, we have developed a TSDF algorithm based on the parallelism properties of CUDA. The target pattern is used as the initial mask in DSO. Using the `CUDA_TSDF` function in Algorithm 2, the initial stage focuses on extracting the boundary segments and computing the distance to the border. We perform pixel-by-pixel `Shift` and `XOR` operations to obtain the mask boundary lines $b_h$ and $b_v$ (lines 2–5). Then, for each pixel $p$ on mask plates, the distance to all boundary lines is computed and the minimal distance between parallel points is determined. Finally, (15) is utilized to construct the TSDF (lines 6–10). Experiment results indicate that `CUDA_TSDF` can reduce TSDF calculation time by more than 98% when applied to a complex mask created by a neural network.

*CUDA-Based Geometry Gradient and Curvature Term:* The numerical settings are compatible with CUDA parallelism, as illustrated by (26). The spatial derivatives of $\phi$ are computed by Algorithm 2 function `CUDA_geometry_gradient`. And the curvature term can be calculated using the GPU-accelerated function `CUDA_curvature`. All the operations in Algorithm 2, such as `shift` and `XOR`, are pixel-wise independent and may be executed per pixel per thread in parallel, which not only reduces the total duration of the DSO but also enables the level set evolution to be integrated into a neural network. As inputs to the `CUDA_curvature` function, we employ the TSDF $\phi$ and geometric gradient $\nabla\phi_x$, $\nabla\phi_y$. We calculate the second-order derivatives in lines 21 and 22 using the `CUDA_geometry_gradient` function. In line 23, we shift the $\phi$ by 1 pixel in four diagonal directions. In line 24, the second-order gradient term $\nabla\phi_{xy}$ is calculated using the central difference method. The curvature term $\kappa$ is finally specified in line 25.

*CUDA-Based Lithography Simulation:* In the ILT-based mask optimization procedure, the lithography simulation will be performed iteratively to guide the contour evolution of the mask. According to the previous experimental research, lithography simulation is the most time-consuming aspect of the mask optimization process because it requires numerous convolution operations between various kernels and mask images. Previous work Neural-ILT [8] transplants the lithography tool-set of the ICCAD 2013 contest onto GPU and develops a high-performance CUDA-based lithography simulation tool [28] to maximize hardware resource utilization and improve computational efficiency. Inspired by Neural-ILT [8], we develop our CUDA-based lithography simulator and incorporate the forward and backward functionalities into the well-known machine learning framework `PyTorch`, along with engineering enhancements. Initially, the optical kernels and accompanying weights are loaded and pinned in GPU memory for the duration of the optimization procedure, ensuring that all computations are conducted on GPU to reduce the data transfer time between CPU and GPU. The `CUDA_FFT` and `CUDA_IFFT` operators are the runtime bottleneck for the CUDA-based lithography simulation. Our enhanced `CUDA_FFT` operator performs quicker than the widely used `cuFFT` and `torch.fft` libraries.

### B. DevelSet-Net

The GPU-accelerated DSO has achieved a great speedup over the level set OPC technique, but there is still considerable opportunity for improvement in terms of runtime and mask printability. We present a novel multibranch transformer-based neural network to improve efficiency and mask printability, inspired by current advances in deep learning-based approaches.

*1) Network Architecture and Training:* The original DSN [31] is a multibranch neural network adopting the simple UNet [32] as the backbone. In this article, we adopt the transformer as the backbone for DSN. As illustrated in Fig. 6, our key contributions include 1) the integration of level set embeddings with the conventional OPC networks; 2) modulation branch for compensating the curvature term defects; and 3) the parameter selector for DSO. DSN achieves an end-to-end trainable deep level set neural network framework for mask optimization.

*CNNs and ViT:* Previous works have argued that CNN is incapable of gathering the required quantity of global information [14]. When compared to CNNs, the level of similarity between the representations is higher in ViT [22], which obtains global representations from shallow layers. Nonetheless, it is necessary to additionally consider the local representations obtained from shallow layers. ViT also adopts skip connections sharing a substantially greater influence than they do in CNN (ResNet), which has a major impact on both the performance and the similarity of the representation. ViT is capable of learning high-quality intermediate representations despite having access to massive volumes of data.

*Vision Transformer Architecture:* As depicted in Fig. 6, the ViT is utilized as the backbone network. The input image will be split into multiple patches of the same height and width $P$

$$x \in \mathbb{R}^{H\times W\times C} \Rightarrow x \in \mathbb{R}^{N\times P^2\times C} \quad (28)$$

where $N = (HW/P^2)$ is the total number of patches, $P$ is the height and width of the patch with a patch dimension $(P, P, C)$, $C$ is the number of channels. The patches are flattened and projected linearly with positional encoding, then fed into the standard transformer. The patch positional encoding is formulated as

$$z_0 = \left[x_{\text{class}}\,; x_p^1E; x_p^2E; \ldots; x_p^NE\right] + E_{\text{pos}} \quad (29)$$

where $E \in \mathbb{R}^{(P^2\cdot C)\ \times D}$, $E_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$. $x_{\text{class}}$ is a learnable embedding to the sequence of patches, at the beginning of the

positional sequence. $E_{\text{pos}}$ is shared 1-D learnable positional parameters that will be added with all patch embeddings to get the resulting input tokens $z_0$.

The transformer layer in Fig. 6 contains the MHA introduced in Section II-D and multilayer perceptrons (MLPs). The residual connections are added after every block

$$\begin{aligned}\mathbf{a_{i-1}} &= \text{MHA}(\text{LN}(\mathbf{z_{i-1}})) + \mathbf{z_{i-1}} \\ \mathbf{z_i} &= \text{MLP}(\text{LN}(\mathbf{a_{i-1}})) + \mathbf{a_{i-1}}\end{aligned} \tag{30}$$

where $i \in \{1, \ldots, L\}$, $L$ is the transformer encoder layer number. LN is layer normalization applied after every block. The decoder in Fig. 6 learns to map the patched embeddings to level set map and modulation map. A point-wise linear layer is applied to produce patch-level mappings and further reshaped into a 2-D feature map.

*Multibranch Pretraining:* To utilize the advance of the multibranch neural networks, two types of losses are optimized simultaneously

$$L_{\text{DSN}}(\theta) = L_0(\theta) + L_m(\theta). \tag{31}$$

*Level Set Branch Supervision:* As illustrated in Fig. 6, different from the typical OPC networks, the level set branch predicts the initial LSF $\phi_{0,\theta}$ for DSO, instead of the pixel-wise mask. This kind of design will preserve the details of the level set function to get a more extensive solution space. And more importantly, the output of DSN can be seamlessly fed into the DSO for further prediction. The mean square error is employed as the objective function

$$L_0(\theta) = \sum_{(x,y)} \left(\phi_{0,\theta}(x, y) - \phi_{\text{gt}}(x, y)\right)^2 \tag{32}$$

where $\phi_{0,\theta}$ is the predicted LSF with network parameters $\theta$. $\phi_{\text{gt}}$ is the ground-truth LSF generated by DSO. Prior to DSN training, the ground truth level set function $\phi_{\text{gt}}$ is obtained by iteratively optimizing the level set function $\phi_0$ using DSO, which serves as the initial LSF, to shape the final mask. This approach has two major advantages: 1) DSN predicts a better initial solution $\phi_{0,\theta}$, which enables DSO to avoid local optima during the optimization process and thus obtain superior optimization results and 2) DSN reduces the number of iterations required by DSO by 90%, thereby significantly reducing the optimization time and achieving higher optimization efficiency.

*Modulation Branch Supervision:* During the training process, the modulation branch aims to find the best $\mathbf{m}_\theta$ in (25) for curvature term evolution in DSO, which is a boundary-aware model for detecting the curvature-sensitive areas. The idea is carried out by shifting the ground-truth TSDF $\phi_{\text{gt}}$ with a set of distance $\Delta h$

$$\begin{aligned}\tilde{\phi}_m(x, y) &= \phi_{\text{gt}}(x, y) + \Delta h \\ \tilde{m} &= H\left(\tilde{\phi}_m(x, y)\right)\end{aligned} \tag{33}$$

where $\Delta h$ is uniformly sampled from $[-20, 20]$, and $\tilde{m}$ is a set of $\mathbf{m}_\theta$. $H(\phi)$ is the Heaviside function

$$H(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0. \end{cases} \tag{34}$$

For every target image $Z_t$, the ground truth of modulation branch is

$$m_{\text{gt}} = \underset{\tilde{m}}{\text{argmin}}\ L_{\text{DSO}}. \tag{35}$$

During the training, the modulation branch learns to simulate an optimized $\mathbf{m}_\theta$. As suggested in [33], the simple Heaviside function in (34) acts on zero level set, which may get stuck in the local minima. To tackle this, we replace it with the approximated Heaviside function (AHF) with a parameter $\varepsilon$

$$H_\varepsilon(\phi) = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{\phi}{\varepsilon}\right)\right). \tag{36}$$

Thus, the objective function is

$$L_m(\theta) = \sum_{(x,y)} \left(H_\varepsilon\left(\phi_{m,\theta}(x, y)\right) - m_{\text{gt}}(x, y)\right)^2 \tag{37}$$

where $H_\varepsilon(\phi_{m,\theta})$ is the output of modulation branch.

### C. Parameter Selector With Transformer

*1) Multitask Learning With Transformer:* Multitask learning has been a promising study topic in numerous domains, including computer vision, natural language processing, multimodal learning, and design space exploration, in order to manage complex optimization tasks. Dedicated models are proposed to address these tasks, either by modeling each activity separately or by employing a holistic probability model to conduct all the tasks simultaneously. It is hypothesized that multitask learning can provide benefits such as enhanced data efficiency, reduced overfitting through shared representations, and rapid learning by exploiting auxiliary information [34].

The majority of prior work on multitask learning focuses on certain domains or modalities with domain-specific model designs. Nonetheless, significant past work exists on multitask learning across domains using a single generic model. It is demonstrated that an encoder–decoder architecture based on the multihead attention mechanism of the transformer can be used to many input and output domains, including picture classification, machine translation, and image captioning. In prior technology, decoders are constructed specifically for each output task, however, our objective is to accomplish simple parameter selection by applying the same decoder architecture to all parameters.

*2) Parameter Selector Algorithm Details:* Due to the huge amount of parameter combinations, however, the advancement of DSO remains difficult. The proposed model for parameter selection is illustrated in Fig. 7. The input images are sent to the image encoder for feature embedding learning. The embeddings and parameter index are then utilized as the decoder's inputs to discover the parameter corresponding to the parameter index. The domain-specific parameter selector in Fig. 6 is Fig. 7. Using the parameter selection, the DSO may automatically alter the lithography and optimization-related parameters according to the supplied designs for optimal results.

*Training of the Parameter Selector:* Before the DevelSet optimization procedure, the suggested parameter selector is trained offline. We set a series of parameter shown in Table II combinations for each image. Then, we use the DevelSet framework without the parameter selector to obtain the optimal parameter group for the input image, which is marked as the

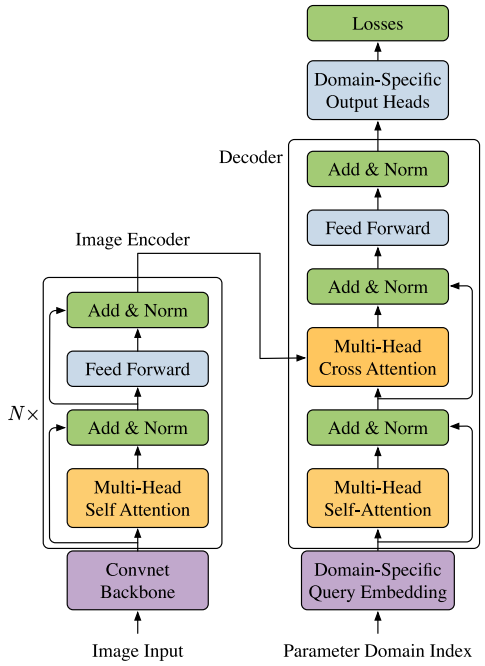


Fig. 7. Domain-specific parameter selector.

TABLE II
PARAMETER SELECTOR COMBINATIONS

| Module | Component | Descriptions | Candidate range |
|---|---|---|---|
| DSO | num_kernel | How many kernels to use | [15, 24] |
| DSO | sigmoid | The sigmoid value in eq. (18) | [25, 80] |
| DSO | pvb_coef | PVB coefficient in eq. (16) | [0.1, 10] |
| DSO | vel_coef | Velocity coefficient in eq. (25) | [0.1, 1] |
| DSO | eta_coef | $\eta$ in CFL | [0.5, 5] |

ground truth. As depicted in Fig. 6, the fine-grained parameter selection will direct the DevelSet Optimizer to achieve optimal results following the training phase.

### D. End-to-End Joint Optimization With DevelSet (DSN+DSO)

As depicted in Fig. 6, the `CUDA_TSDF` function is utilized to enable the quick transformation from pixel-wise target image to LSF $\phi_0$. After pretraining the two branches of DSN, we fix all the parameters of DSN and feed the output of the level set branch $\phi_{0,\theta}$, modulation branch $\mathbf{m}_\theta$, and domain-specific parameter selector directly into the evolution process of DSO to build the final mask. We use the CG method [19] for optimization in DSO, and we use the CFL condition [19] to determine the time step $\Delta t = \eta/\max(|v|)$, where $v$ is the evolution velocity in (22), and $\eta$ is CFL condition number.

### E. Adaption to Large-Scale Layout

Large-scale mask optimization poses new challenges due to the fact that industrial cases are often much larger than those commonly encountered in academic research. To address this issue, we draw on the large tile global perception algorithm proposed by [14], and further optimize the stitching problems and redesign the scheduling to optimize the use of GPU memory. Due to the restriction of the optical diameter,

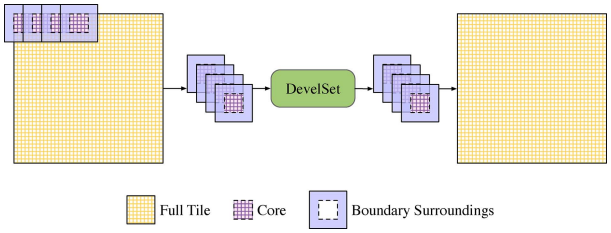


Fig. 8. Algorithm for applying the DevelSet framework to large-size layouts involves dividing the large layout into multiple small blocks and obtaining optimized results for each block. Each block is composed of a core part and boundary surroundings.

as discussed in [14], the large tile must first be cut into smaller clips. These clips are then optimized using the DevelSet framework in a sliding-window manner. As depicted in Fig. 8, each window is divided into a core part and a boundary surrounding. The latter item is specifically devised to minimize boundary distortion effects resulting from lithographic physics to the greatest extent possible. This is achieved by progressively encompassing the entire layout with the core parts through sliding-window movement, while the boundary parts are disregarded, thus enhancing fidelity. The optimized masks covered by the core parts are then concatenated back to the large-scale tile as the final result.

## IV. EXPERIMENTAL RESULTS

### A. Experimental Setup

*1) Implementation Details:* We implement the proposed DevelSet-TFPS flow under the `PyTorch` framework. The ViT backbone is from [22]. The dataset for pretraining DSN is obtained from PGAN-OPC [6], which contains 4875 images of $2048 \times 2048$-pixel images generated following the same design rule of ICCAD 2013 contest [28]. Ten test cases are also from [28] with industrial M1 designs and 32 nm design node. The lithography engine is from [28], with 193-nm wavelength, a defocus range of $\pm 25$ nm, and a dose range of $\pm 2\%$. Before the end-to-end joint optimization, we use GPU-accelerated DSO to generate quasi-optimized level set ground truth for DSN pretraining. The shot count metric is evaluated using scripts from [8] to ensure a fair comparison.

The DevelSet framework is further evaluated on large-scale datasets in this study. Specifically, the large layouts are generated from the FreePDK45 [35] design kit using OpenROAD [36] without loss of generality. Ten dense 144 $\mu$m$^2$ layouts were randomly tiled and copied from `layer 9` of FreePDK45 [35] with manual density adjustments.

The resulting layouts are then converted to $12\,000 \times 12\,000$-pixel images. Seven of these large layouts are used as the training set, while the remaining three are used as the test set. In the evaluation, the size of the sliding-window depicted in Fig. 8 is set to be $2048 \times 2048$, with the core part having $1024 \times 1024$ pixels.

*2) Training and Testing Environment:* We use 8 Nvidia RTX 3090 GPU for DSN pretraining and a single Nvidia RTX 3090 for joint optimization. The complete training process of DSN, which includes the level set branch, modulation branch, and parameter selector, takes 31.5 h. Throughout this article, we pick $\sigma_z = 50$, $N_h = 24$, $\alpha = 1$, $\beta = 7.5$, $\lambda = 0.9$,

TABLE III
MASK PRINTABILITY, COMPLEXITY COMPARISON WITH SOTA

| Bench | Area($nm^2$) | PGAN-OPC [6] | | | GLS-ILT [21] | | | Neural-ILT [8] | | | DevelSet [32] | | | DevelSet-TFPS | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $L_2$ | PVB | #shots | $L_2$ | PVB | #shots | $L_2$ | PVB | #shots | $L_2$ | PVB | #shots | $L_2$ | PVB | #shots |
| case1 | 215344 | 52570 | 56267 | 931 | 46032 | 62693 | 1476 | 50795 | 63695 | 743 | 49142 | 59607 | 969 | 48687 | 57612 | 917 |
| case2 | 169280 | 42253 | 50822 | 692 | 36177 | 50642 | 861 | 36969 | 60232 | 571 | 34489 | 52012 | 743 | 34062 | 49647 | 731 |
| case3 | 213504 | 83663 | 94498 | 1048 | 71178 | 100945 | 2811 | 94447 | 85358 | 791 | 93498 | 76558 | 889 | 92025 | 76744 | 847 |
| case4 | 82560 | 19965 | 28957 | 386 | 16345 | 29831 | 432 | 17420 | 32287 | 209 | 18682 | 29047 | 376 | 18194 | 27318 | 333 |
| case5 | 281958 | 44733 | 59328 | 950 | 47103 | 56328 | 963 | 42337 | 65536 | 631 | 44256 | 58085 | 902 | 42158 | 57123 | 891 |
| case6 | 286234 | 46062 | 52845 | 836 | 46205 | 51033 | 942 | 39601 | 59247 | 745 | 41730 | 53410 | 774 | 41611 | 51096 | 754 |
| case7 | 229149 | 26438 | 47981 | 515 | 28609 | 44953 | 548 | 25424 | 50109 | 354 | 25797 | 46606 | 527 | 24265 | 43032 | 493 |
| case8 | 128544 | 17690 | 23564 | 286 | 19477 | 22541 | 439 | 15588 | 25826 | 467 | 15460 | 24836 | 493 | 15301 | 22131 | 436 |
| case9 | 317581 | 56125 | 65417 | 1087 | 52613 | 62568 | 881 | 52304 | 68650 | 653 | 50834 | 64950 | 932 | 49843 | 62139 | 887 |
| case10 | 102400 | 9990 | 19893 | 338 | 22415 | 18769 | 333 | 10153 | 22443 | 423 | 10140 | 21619 | 393 | 10341 | 20104 | 398 |
| Average | | 39948.9 | 49957.2 | 706.9 | 38615.4 | 50030.3 | 968.6 | 38503.8 | 53338.3 | **558.7** | 38402.8 | 48673.0 | 699.8 | **37648.7** | **46694.6** | 668.7 |
| Ratio | | 1.061 | 1.070 | 1.057 | 1.026 | 1.071 | 1.448 | 1.023 | 1.142 | **0.836** | 1.020 | 1.042 | 1.047 | **1.000** | **1.000** | 1.000 |

† *$L_2$ and PVB unit: $nm^2$.*

$\eta = 0.85$, and $\varepsilon = 0.03$, as default parameters for DevelSet optimization.

### B. Comparison With State-of-the-Art

We first compare the performance of DevelSet with SOTA works, including PGAN-OPC [6], GLS-ILT [21], and Neural-ILT [8] in Table III. The "$L_2$," "PVB," and "#shots" are introduced in Section (II-C), representing *Squared $L_2$ error*, *PVBand*, and *Mask fracturing shot count*, respectively. DevelSet-TFPS refers to DevelSet [31] with transformer-based parameter selector.

*1) Mask Printability and Complexity:* As illustrated in Table III, compared with previous SOTA works, DevelSet-TFPS can achieve better performance on $L_2$ and PVB metrics, demonstrating superior mask printability. Compared with learning-based methods PGAN-OPC and Neural-ILT, DevelSet-TFPS can achieve 6.1%, 2.3% better $L_2$ and 7%, 14.2% better PVB. In addition, DevelSet-TFPS also outperforms the previous level set-based method GLS-ILT with improvements of 2.6% on $L_2$ and 7.1% on PVB, respectively.

The #shots metric is capable of measuring the mask complexity to a limited degree. Among the PGAN-OPC, GLS-ILT, DevelSet-TFPS can reduce 5.7%, 44.8% #shots. Neural-ILT, which adds an additional branch for complexity reduction, can achieve a smaller #shots number than DevelSet-TFPS. We visualize the samples of results in Fig. 9. As depicted in Fig. 9, our framework has fewer isolated stitches and its boundaries are smoother. Considering the implementation of #shots visualized in Fig. 10, it will cut the whole mask into fragmented small rectangles, which means, if the edges are flatter, then there will be fewer shots. In fact, this is inconsistent with the industrial case. Therefore, shots are only a point of reference for the level of complexity and are not a deciding element.

Table IV presents the EPE results for the DevelSet-TFPS method and other methods on the Benchmark dataset. The DevelSet-TFPS method achieves an average EPE value of 7.3, which is the lowest among all the learning-based methods. The performance loss in case3 is mainly caused by the model first ensuring the optimization of PVB and L2. However, in most cases, such as case1-2 and case4-10, our approach achieves better EPE values. In comparison, the previous GLS-ILT method [21] adds an additional penalty for EPE performance, resulting in better EPE values than our method. Overall, our results demonstrate the significance of continued research and development of level set-based methods in the field of mask optimization and the improvement that our approach brings to the existing literature.

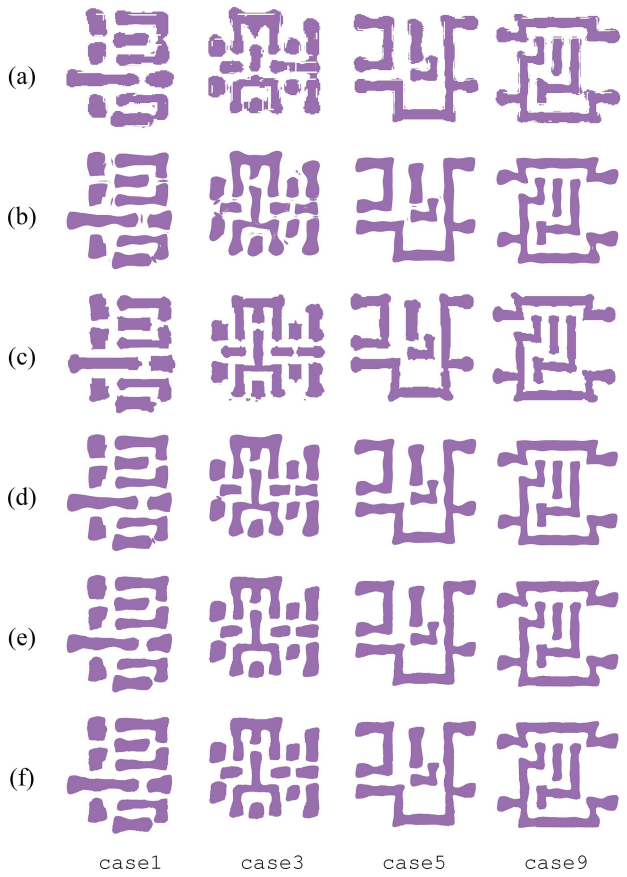


Fig. 9. Mask visualizations of: (a) PGAN-OPC [6], (b) GLS-ILT [21], (c) Neural-ILT [8], (d) DSO, (e) DevelSet framework (DSN + DSO), and (f) DevelSet-TFPS: DevelSet framework with transformer-based parameter selector.

*2) Runtime Comparison:* We list the runtime comparison in Table V, where the metric "TAT" refers to turn around time,

TABLE IV
EPE COMPARISON WITH SOTA

| Bench | PGAN [6] EPE | GLS [21] EPE | NILT [8] EPE | DevelSet [32] EPE | DevelSet-TFPS EPE |
|---|---|---|---|---|---|
| case1 | 8 | 4 | 8 | 10 | 8 |
| case2 | 13 | 1 | 3 | 1 | 1 |
| case3 | 51 | 29 | 52 | 64 | 59 |
| case4 | 2 | 0 | 2 | 2 | 2 |
| case5 | 8 | 0 | 3 | 1 | 1 |
| case6 | 12 | 0 | 5 | 2 | 2 |
| case7 | 7 | 0 | 0 | 0 | 0 |
| case8 | 0 | 0 | 0 | 0 | 0 |
| case9 | 12 | 1 | 2 | 0 | 0 |
| case10 | 0 | 0 | 0 | 0 | 0 |
| Average | 11.30 | 3.5 | 7.5 | 8.0 | 7.3 |
| Ratio | 1.413 | 0.438 | 0.938 | 1 | 0.913 |

TABLE V
RUNTIME COMPARISON WITH SOTA

| Bench | PGAN [6] TAT (s) | GLS [21] TAT (s) | NILT [8] TAT (s) | DevelSet [32] TAT (s) | DevelSet-TFPS TAT (s) |
|---|---|---|---|---|---|
| case1 | 358 | 123 | 13.57 | **1.5** | 1.91 |
| case2 | 368 | 81 | 14.37 | **1.4** | 1.83 |
| case3 | 368 | 214 | 9.72 | **1.29** | 1.76 |
| case4 | 377 | 184 | 10.4 | **1.65** | 2.21 |
| case5 | 369 | 76 | 10.04 | **0.91** | 1.34 |
| case6 | 364 | 65 | 11.11 | **0.84** | 1.37 |
| case7 | 377 | 64 | 9.67 | **0.76** | 1.52 |
| case8 | 383 | 67 | 11.81 | **1.14** | 1.63 |
| case9 | 383 | 63 | 9.68 | **1.21** | 1.77 |
| case10 | 366 | 64 | 11.46 | **0.42** | 1.0 |
| Average | 371.3 | 100.1 | 11.18 | **1.11** | 1.63 |
| Ratio | 334.505 | 90.180 | 10.072 | **1.000** | 1.468 |

calculating the total interval for the end-to-end inference in seconds (s). PGAN-OPC uses the traditional mask optimizer from MOSAIC [4] for result fine-tuning, thus our framework can achieve ∼300× acceleration than it. GLS-ILT applies GPU to speed up the FFT calculation in the mask optimization process, but increases the memory exchange time between CPU and GPU, which makes it ∼100× slower than our framework. Compared with SOTA machine learning-based method Neural-ILT, that regards the optimization procedure as network training, our framework can boost runtime performance ∼10× By replacing the backbone to transformer, DevelSet-TFPS improve the mask printability at a little expense on runtime performance, with an average of 1.63 s (DevelSet 1.11 s).

### C. Ablation Study

We also conduct a series of ablation studies to verify the effectiveness of each component of DevelSet. In Table VI, we evaluate the influence of curvature term and modulation branch on DSO and DSN. The column "w/o. curv." refers to the experiments without curvature term and "w/o. mod." refers to the experiments without modulation branch. In Table VII, we measure the effect of backbone and the proposed parameter selector.

TABLE VI
ABLATION STUDY ON DSN AND DSO

| | DSO | | DSN+DSO | |
|---|---|---|---|---|
| | w/o. curv. | w. curv. | w/o. mod. | w. mod. |
| $L_2$ | 38253.0 | 38454.0 | 39259.8 | 38402.8 |
| PVB | 49243.0 | 49398.0 | 48384.0 | 48673.0 |
| #shots | 805.0 | 726.0 | 712.8 | 699.8 |
| cost† | 95546.0 | 95112.0 | 94771.8 | **94073.8** |

†*cost* = $L_2$ + *PVB* + 10 × *#shots*.

TABLE VII
EXPERIMENT ON BACKBONE AND PARAMETER SELECTOR

| | DevelSet (CNN) | DevelSet-TF | DevelSet-TFPS |
|---|---|---|---|
| $L_2$ | 38402.8 | 38297.0 | 37648.7 |
| PVB | 48673.0 | 48059.0 | 46694.6 |
| #shots | 699.8 | 682.1 | 668.7 |
| cost† | 94073.8 | 93177.0 | **91030.3** |

†*cost* = $L_2$ + *PVB* + 10 × *#shots*.

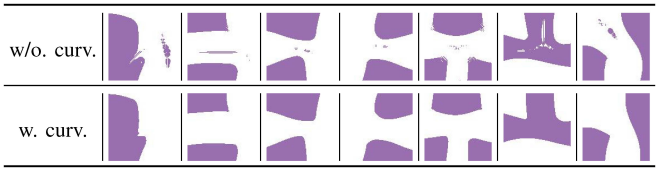


Fig. 10. Visualizations for ablation study of the curvature term.

*1) Effectiveness of Curvature Term:* As shown in Table VI, both DSO and DevelSet are influenced by the curvature term. The cost is calculated as

$$\text{cost} = L_2 + \text{PVB} + 10 \times \#\text{shots}. \tag{38}$$

The results are better if the cost is smaller. The total cost has been reduced 434 on DSO by curvature term with fewer #shots The $L_2$ and PVB increase a bit due to the tradeoff between the mask printability and complexity, which motivates us to use modulation branch to mitigate the negative impact.

*2) Necessity of DSN and Modulation Branch:* Considering the effect of modulation branch and DSN in Table VI, modulation branch reduces the total cost by 698 showing superior improvements on mask printability. DSN can generate better initial solution for DSO to boost the overall performance of the whole framework, which reduces the cost by 1038.2. The improvement can be attributed to that DSN can help the framework overcome the local minima and obtain a better mask. DSN also contributes to fewer #shots because the upsample operation in neural network allows for straighter borders. At the same time, DSN accelerates the mask optimization process using the fast inference ability of neural networks. The modulation branch is similar to the attention mechanism, which sets the area for the curvature term to influence the overall optimization process.

*3) Ablation Study on Backbone and Parameter Selector:* As shown in Table III, the last column presents the $L_2$, PVB and #shots comparisons of the proposed parameter selector. By replacing the CNN backbone to transformer, DevelSet with transformer-based parameter selector surpasses the original

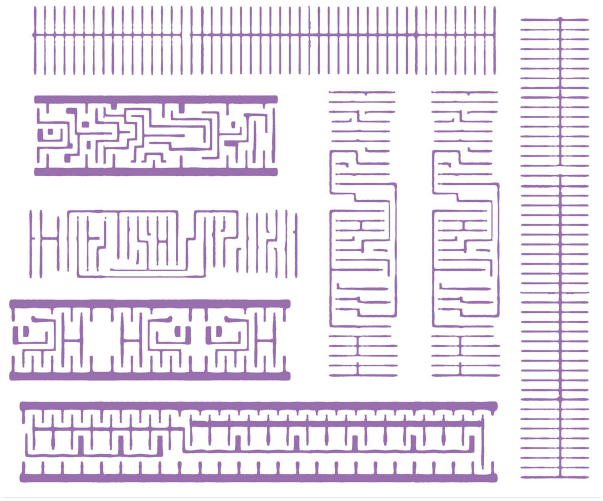

Fig. 11. Results of applying the DevelSet framework to large-scale layouts.

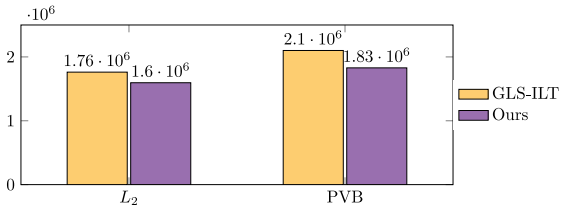


Fig. 12. $L_2$ and PVB comparison of large tile layout between GLS-ILT [21] and DevelSet-TFPS.

DevelSet by 2% and 4.1% in $L_2$ and PVB, respectively. We also conduct a series of ablation studies on the parameter selector in Table VII. The transformer backbone leveraging attention mechanism improves the performance and the overall cost reduces by 896.8. And the parameter selector boosts the performance to a large margin with 3043.5 cost deduction.

*4) Evaluation on Large-Scale Layout:* Fig. 11 illustrates an example of a large-scale layout optimized using DevelSet, consisting of a 144 $\mu m^2$ area. The layout is divided into small windows of size $2048 \times 2048$ pixels, containing a core region of $1024 \times 1024$ pixels and boundary surroundings, using the algorithm described in Section III-E. The optimized core regions are then concatenated to form the final large-scale optimized layout. The results demonstrate the effectiveness of DevelSet in avoiding stitching errors in large-scale layouts. Through conducting experiments on three large-scale test layouts, we have compared the average performances of level set-based methods, GLS-ILT [21] and DevelSet-TFPS, in terms of $L_2$ and PVB metrics. The results presented in Fig. 12 demonstrate that DevelSet-TFPS achieves a reduction of 10% and 12.9% in $L_2$ and PVB, respectively, compared to GLS-ILT [21]. These results highlight the potential of DevelSet for large-scale layout optimization in the field of mask optimization.

## V. Conclusion

In this article, we propose an implicit level set-based evolution framework for mask optimization, abandoning the discrete pixel representation. The implicit representation makes it easier to control boundaries precisely, and we propose a curvature term to regulate boundary shapes, improving manufacturability. We perform all calculations, including level set initialization, curvature term, level set evolution, and mask optimization, on the GPU to speed up the process. We design a multibranch transformer-based network to predict a better initialization of the level set function $\phi$ and an attention-like matrix from the modulation branch to compensate for the performance loss caused by the curvature term. Additionally, the parameter selection simplifies the manual parameter setting process required for optimal condition optimization. Our experimental results show that the proposed DevelSet framework achieves a 2% and 4% reduction in $L_2$ and PVBand, respectively, with superior runtime performance. We expect this enhanced level set technique with CUDA/DNN-accelerated joint optimization paradigm to have a significant impact on industrial mask optimization solutions.

## References

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

[2] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama, "A fast process variation and pattern fidelity aware mask optimization algorithm," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2014, pp. 238–245.

[3] 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.

[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] 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.

[6] 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.

[7] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[8] B. Jiang, L. Liu, Y. Ma, H. Zhang, B. Yu, and E. F. Young, "Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

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

[10] S. Choi, S. Shim, and Y. Shin, "Neural network classifier-based OPC with imbalanced training data," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 938–948, May 2019.

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

[12] Q. Wang, B. Jiang, M. D. F. Wong, and E. F. Y. Young, "A2-ILT: GPU accelerated ILT with spatial attention mechanism," in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 967–972.

[13] B. Jiang, X. Zhang, L. Liu, and E. F. Young, "Building up end-to-end mask optimization framework with self-training," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2021, pp. 63–70.

[14] H. Yang et al., "Generic lithography modeling with dual-band optics-inspired neural networks," in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 973–978.

[15] H. Yang and H. Ren, "Enabling scalable AI computational lithography with physics-inspired models," in *Proc. 28th Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2023, pp. 715–720.

[16] J. A. Sethian and D. Adalsteinsson, "An overview of level set methods for etching, deposition, and lithography development," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 1, pp. 167–184, Feb. 1997.

[17] Y. Shen, N. Wong, and E. Y. Lam, "Level-set-based inverse lithography for photomask synthesis," *Opt. Exp.*, vol. 17, no. 26, pp. 23690–23701, Dec. 2009.
[18] Y. Shen, N. Jia, N. Wong, and E. Y. Lam, "Robust level-set-based inverse lithography," *Opt. Exp.*, vol. 19, no. 6, pp. 5511–5521, 2011.
[19] 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," *J. Vac. Sci. Technol. B*, vol. 31, no. 4, Jul. 2013, Art. no. 41605.
[20] Z. Geng, Z. Shi, X.-L. Yan, K.-S. Luo, and W.-W. Pan, "Fast level-set-based inverse lithography algorithm for process robustness improvement and its application," *J. Comput. Sci. Technol.*, vol. 30, no. 3, pp. 629–638, 2015.
[21] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. IEEE/ACM Design Autom. Test Eurpoe (DATE)*, 2020, pp. 1835–1838.
[22] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. 9th Int. Conf. Learn. Rep. (ICLR)*, May 2021, pp. 1–6.
[23] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, p. 10.
[24] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 213–229.
[25] M. Chen et al., "Generative pretraining from pixels," in *Proc. 37th Int. Conf. Mach. Learn.*, vol. 119, Jul. 2020, pp. 1691–1703. [Online]. Available: https://proceedings.mlr.press/v119/chen20s.html
[26] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, 1988.
[27] H. Hopkins, "The concept of partial coherence in optics," *Proc. Roy. Soc. London A Math. Phys. Eng. Sci.*, vol. 208, no. 1093, pp. 263–277, 1951.
[28] 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.
[29] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, "Uniformly high order accurate essentially non-oscillatory schemes, III," *J. Comput. Phys.*, vol. 71, no. 2, pp. 231–303, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021999187900313
[30] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci. USA*, vol. 93, no. 4, pp. 1591–1595, 1996.
[31] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2021, pp. 1–9.
[32] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.*, 2015, pp. 234–241.
[33] T. F. Chan and L. A. Vese, "Active contours without edges," *IEEE Trans. Image Process.*, vol. 10, no. 2, pp. 266–277, Feb. 2001.
[34] M. Crawshaw, "Multi-task learning with deep neural networks: A survey," 2020, *arXiv:2009.09796*.
[35] "NanGate FreePDK45 generic open cell library." 2008. [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk45/
[36] T. Ajayi and D. Blaauw, "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. Govt. Microcircuit Appl. Crit. Technol. Conf.*, 2019, pp. 1–9.

**Guojin Chen** received the B.Eng. degree in software engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His current research interests include machine learning in VLSI design for manufacturability and physics-informed networks for solving EDA area problems.

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

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

**Hongduo Liu** received the B.E. degree from the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

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

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

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

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

Dr. Yu received nine Best Paper Awards from DATE 2022, ICCAD 2021 and 2013, ASPDAC 2021 and 2012, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, 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.