2026 Design, Automation & Test in Europe Conference (DATE 2026)

# Scalable Second-Order Optimizer for Full-Chip Inverse Lithography Techniques

Su Zheng[1], Ziyang Yu[1,#], Bei Yu[1,#], Martin Wong[2]
[1]Chinese University of Hong Kong [2]Hong Kong Baptist University

***Abstract*—Full-chip inverse lithography techniques (ILT) represent an advanced methodology for next-generation mask optimization, enhancing sub-wavelength patterning but often facing prohibitive computational costs. State-of-the-art methods rely on iterative first-order optimizers, which exhibit slow convergence, often requiring hundreds of iterations. This inefficiency is compounded by the high overhead of repeated Fast Fourier Transform (FFT) operations and inter-GPU communication per iteration. To overcome this fundamental bottleneck, we propose a scalable second-order optimizer for full-chip ILT. Our approach leverages second-order curvature information via the Hessian matrix to achieve dramatically faster convergence and superior pattern fidelity compared to conventional first-order methods. Crucially, we address the prohibitive cost of exact Hessian computation by employing Hutchinson's method to efficiently approximate the Hessian diagonal. Combined with exponential moving average (EMA) and gradient modulation techniques, our optimizer achieves significant performance gains. Experimental results demonstrate substantial improvements in both runtime efficiency (reduced iterations) and solution quality (enhanced pattern fidelity) compared to existing first-order ILT methods, paving the way for practical full-chip ILT.**

## I. INTRODUCTION

Semiconductor manufacturing is the foundation of modern computing, enabling increasingly sophisticated applications across edge devices and hyperscale datacenters. At the heart of this process is lithography, the primary technique for transferring circuit patterns from photomasks onto silicon wafers. As technology nodes shrink beyond the wavelength limits of lithography systems, optical diffraction introduces severe pattern distortions that threaten both yield and reliability [1]. To mitigate these effects, computational lithography has become essential, ensuring that printed patterns accurately reproduce the intended circuit designs [2].

Mask optimization techniques [1]–[11] encompass rule-based and model-based optical proximity correction (OPC), sub-resolution assist features (SRAFs), and inverse lithography techniques (ILT). Rule-based OPC [12]–[14] relies on handcrafted geometric heuristics, while model-based OPC [15]–[21] uses lithography simulations to iteratively refine mask shapes. ILT [22]–[28] represents a paradigm shift, casting mask optimization as an inverse imaging problem. By employing gradient-based optimization, ILT can jointly refine main features and SRAFs with multiple objectives. This unified approach yields significantly higher pattern fidelity compared to the disjoint workflows of traditional OPC, particularly at advanced nodes.

Scaling ILT to full-chip layouts remains a major challenge. Academic methods [29] often divide layouts using geometric or pattern-based heuristics, while industrial approaches [30] employ overlapping tile strategies followed by heuristic post-optimization healing. These techniques are moderately effective for via layers but struggle with complex metal layers, where strong inter-tile optical interactions dominate. Recent efforts, such as gradient healing [31], aim to reduce boundary artifacts by coordinating global gradient updates.

Despite these progresses, full-chip ILT deployment faces prohibitive computational barriers. Treating billion-pixel masks as optimization variables creates massive parameter spaces, while partitioning strategies introduce boundary artifacts that degrade pattern quality [30]. Current state-of-the-art methods rely heavily on first-order optimization, which converges slowly on ill-conditioned loss landscapes. Each iteration incurs significant computational cost due to Fast Fourier Transforms (FFTs) required for simulation and gradient computation. Moreover, distributed implementations suffer from inter-GPU communication overhead during gradient synchronization [31], making full-chip ILT impractically slow.

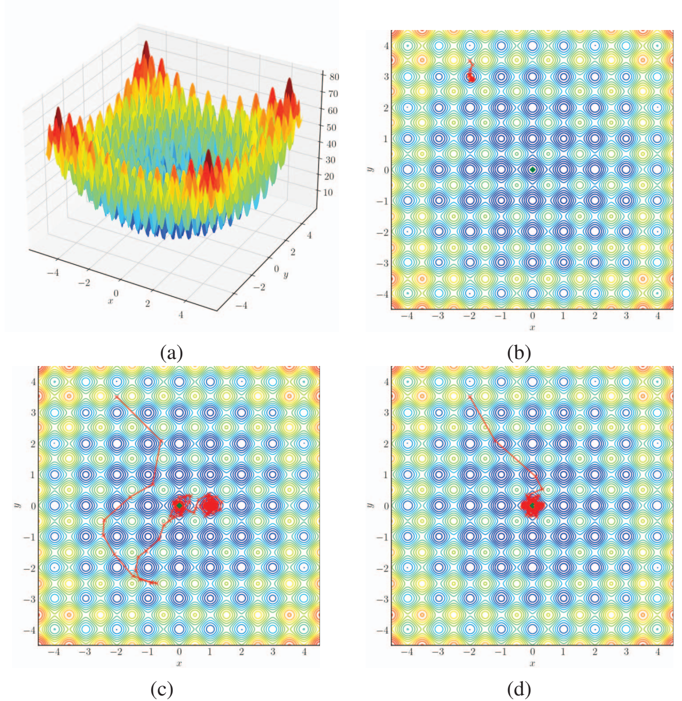


Fig. 1 Visualization of (a) Rastrigin function (global minimum at $(0,0)$) and comparison of convergence speed between first-order optimizers (b) gradient descent, (c) Adam [32], and (d) a second-order optimizer AdaHessian [33]. The gradient descent optimizer converges at local minimum. The second-order optimizer converges to global optimal significantly faster than the Adam optimizer, demonstrating superior efficiency in complex problems.

To address these bottlenecks, we propose a scalable second-order optimizer tailored for full-chip ILT. As illustrated in Fig. 1, second-order optimizers significantly outperform first-order methods in terms of convergence speed. Building on this insight, our method leverages curvature information via the Hessian matrix to accelerate convergence and improve pattern fidelity over first-order baselines. To avoid the prohibitive cost of computing the full Hessian, we use Hutchinson's stochastic estimator to efficiently approximate its

[#] *Corresponding authors: {zyyu21, byu}@cse.cuhk.edu.hk*

978-3-9826741-1-7/DATE26/© 2026 EDAA

diagonal. We further enhance stability and robustness by incorporating exponential moving averages (EMA) and gradient modulation. The resulting optimizer achieves significantly faster convergence with minimal overhead, enabling improved pattern fidelity and runtime efficiency. Experiments on industry-scale layouts demonstrate that our approach can reduce iteration counts by 75% with improved pattern fidelity. These improvements propagate throughout the full-chip flow, offering a practical and scalable solution.

The major contributions of this paper are summarized as follows:

- We introduce a second-order optimization framework specifically designed for full-chip ILT, addressing the limitations of conventional first-order approaches.
- To avoid the high computational cost of full Hessian computation, our optimizer employs Hutchinson's stochastic estimator for efficient diagonal approximation.
- We integrate exponential moving averages (EMA) and gradient modulation techniques to improve convergence speed and robustness of full-chip ILT.
- Extensive evaluations on real-world full-chip layouts demonstrate substantial improvements over state-of-the-art methods, making full-chip ILT viable for industrial applications.

The remainder of this paper is organized as follows: Section II provides background on full-chip ILT. Section III details our scalable second-order optimizer for full-chip ILT. Section IV presents the experimental results that can demonstrate the effectiveness of our method, and Section V concludes this paper.

## II. Preliminaries

### A. Inverse Lithography Techniques

ILT algorithms usually optimize unconstrained parameters $\boldsymbol{x} \in \mathbb{R}^{N\times N}$, which are transformed to the mask image $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ via:

$$\boldsymbol{M}(x,y) = \sigma_M(\boldsymbol{x}(x,y)) = \frac{1}{1+\exp\left(-\theta_M \boldsymbol{x}(x,y)\right)}, \quad (1)$$

where $\theta_M$ controls the steepness of the sigmoid function. This transformation limits the values of $\boldsymbol{M}$ to the range $[0,1]$.

The pattern transfer from a mask to a wafer is modeled using lithography simulation, which typically comprises two main components: the optical projection model and the photoresist model. In the optical projection stage, the mask image $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ is transformed into an aerial image $\boldsymbol{I} \in \mathbb{R}^{N\times N}$, representing the light intensity distribution on the wafer surface. This transformation is commonly described by the Hopkins diffraction model [34], which can be approximated using the Sum of Coherent Systems (SOCS) formulation:

$$\boldsymbol{I} = \sum_{k=1}^{K} \mu_k \left|\boldsymbol{H}_k \otimes \boldsymbol{M}\right|^2, \quad (2)$$

where $\boldsymbol{H}_k \in \mathbb{C}^{N\times N}$ denotes the $k$-th optical kernel, $\mu_k$ is the corresponding weight, and $\otimes$ represents the convolution operation.

The resulting aerial image $\boldsymbol{I}$ is then passed through a photoresist model to produce the final printed image $\boldsymbol{Z}$. To support gradient-based optimization in ILT, a differentiable sigmoid-based approximation is commonly applied to each pixel in the aerial image:

$$\boldsymbol{Z}(x,y) = \sigma_Z(\boldsymbol{I}(x,y)) = \frac{1}{1+\exp(-\theta_Z(\boldsymbol{I}(x,y) - I_{th}))}, \quad (3)$$

where $I_{th}$ is the intensity threshold, and $\theta_Z$ controls the sharpness.

To ensure robustness under process variations, ILT methods typically simulate multiple process corners using optical kernels corresponding to maximum, nominal, and minimum conditions, producing printed images $\boldsymbol{Z}_{\text{max}}$, $\boldsymbol{Z}_{\text{nom}}$, and $\boldsymbol{Z}_{\text{min}}$, respectively.

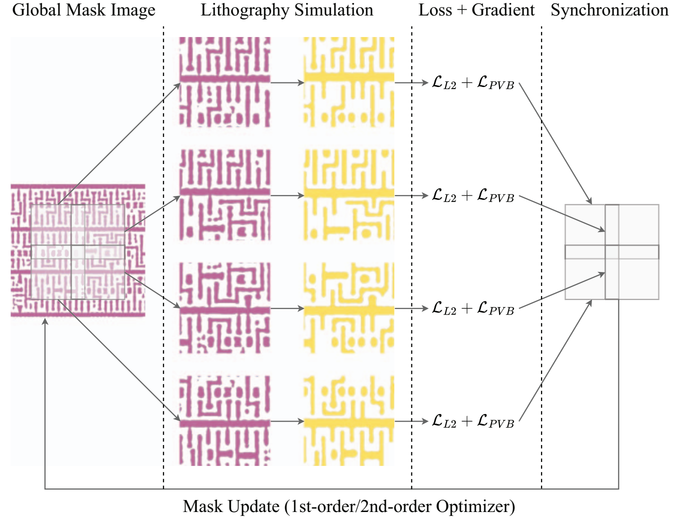


Fig. 2 Illustration of a typical full-chip ILT flow. At each tile, the global mask is partitioned into overlapping tiles to facilitate parallel processing. Each tile undergoes lithography simulation to generate the corresponding printed patterns. Subsequently, loss functions are evaluated, and gradients from all tiles are aggregated through global synchronization to update the global mask.

The optimization objective usually includes the L2 loss between the nominal printed image $\boldsymbol{Z}_{\text{nom}}$ and the design target $\boldsymbol{Z}_Y$:

$$\mathcal{L}_{L2}(\boldsymbol{Z}_{\text{nom}}, \boldsymbol{Z}_Y) = \left\|\boldsymbol{Z}_{\text{nom}} - \boldsymbol{Z}_Y\right\|^2, \quad (4)$$

as well as the process variation band (PVB) loss, which quantifies sensitivity to process fluctuations:

$$\mathcal{L}_{PVB}(\boldsymbol{Z}_{\text{max}}, \boldsymbol{Z}_{\text{min}}) = \left\|\boldsymbol{Z}_{\text{max}} - \boldsymbol{Z}_{\text{min}}\right\|^2. \quad (5)$$

These losses are standard metrics for evaluating the fidelity and robustness of optimized masks. For evaluation, printed images are binarized using a threshold of 0.5.

At each ILT iteration, the gradient $\frac{\partial\mathcal{L}}{\partial\boldsymbol{x}}$ is computed based on the total loss function $\mathcal{L} = \mathcal{L}_{L2} + \mathcal{L}_{PVB}$. Using a gradient descent optimizer, the parameters $\boldsymbol{x}$ are updated according to $\boldsymbol{x} \leftarrow \boldsymbol{x} - \gamma\frac{\partial\mathcal{L}}{\partial\boldsymbol{x}}$, where $\gamma$ denotes the learning rate. Once optimization converges, the final parameters $\boldsymbol{x}^*$ are used to generate the optimized mask $\boldsymbol{M}^*$.

In addition to L2 and PVB, edge placement error (EPE) is also assessed to measure pattern fidelity. EPE is computed by sampling probe points along target edges and counting those where the printed edge deviates from the target by more than a predefined threshold. For further details on these metrics, please refer to [23].

### B. Full-Chip ILT

To manage the complexity of full-chip ILT, current implementations adopt tiling strategies, where the layout is partitioned into smaller regions that are optimized independently or with limited coordination. While tiling allows for parallelization and reduces memory demands, it introduces new challenges such as stitching artifacts and boundary inconsistencies due to the inherently nonlocal nature of optical interactions. These artifacts become particularly pronounced in metal layers containing dense, elongated patterns.

To minimize stitching artifacts, FuILT [31] proposes a synchronous optimization paradigm for full-chip ILT. Fig. 2 illustrates the standard FuILT workflow, demonstrating how large-scale mask optimization

Fig. 3 Illustration of the gradient vector and Hessian matrix. The visualization highlights the diagonal dominance of the Hessian matrix in ILT, which inspires us to precondition the gradient using the inverse of Hessian matrix's diagonal.

becomes tractable through spatial decomposition and globally synchronized updates. The process begins by partitioning the full-chip mask layout into overlapping tiles, allowing for localized lithography simulations and parallel processing. Each tile is independently simulated using the Hopkins model to estimate the corresponding printed patterns. Loss functions are then computed to capture both the deviation from the target pattern and the discrepancies in printability across different process conditions. Gradients from all tile are calculated and globally synchronized to ensure consistent updates across overlapping regions. The coordinated gradient aggregation enables scalable optimization while effectively reducing stitching artifacts.

## III. Method

Our full-chip ILT method builds upon the FuILT algorithm [31], which is introduced in Section II-B. We enhance the original framework by replacing its first-order optimizer with the proposed second-order optimizer (Section III-A), which incorporates Hessian matrix approximation (Section III-B), exponential moving averages (Section III-C), and gradient modulation (Section III-D) techniques.

### A. Second-Order Optimizer

Based on the discussion in Section II-A, the ILT problem can be formulated as the minimization of a loss function:

$$\min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}). \tag{6}$$

Without loss of generality, we can consider the parameters as a vector $\boldsymbol{x} = [x_1, x_2, \cdots, x_{N^2}]^T$. To analyze the optimization landscape, the loss function $\mathcal{L}(\boldsymbol{x})$ can be approximated using Taylor expansion:

$$\mathcal{L}(\boldsymbol{x}+\Delta\boldsymbol{x}) \approx \mathcal{L}(\boldsymbol{x})+\Delta\boldsymbol{x}^T\nabla\mathcal{L}(\boldsymbol{x})+\frac{1}{2}\Delta\boldsymbol{x}^T\left(\nabla^2\mathcal{L}(\boldsymbol{x})\right)\Delta\boldsymbol{x}+\epsilon(\boldsymbol{x}), \tag{7}$$

where $\nabla\mathcal{L}(\boldsymbol{x})$ is the gradient, $\nabla^2\mathcal{L}(\boldsymbol{x})$ is the Hessian matrix, and $\epsilon(\boldsymbol{x})$ denotes higher-order terms.

To derive the optimal update direction $\Delta\boldsymbol{x}$, we can ignore the higher-order terms and set the gradient of the Taylor approximation with respect to $\Delta\boldsymbol{x}$ to zero:

$$\Delta\boldsymbol{x} = -\left(\nabla^2\mathcal{L}(\boldsymbol{x})\right)^{-1}\nabla\mathcal{L}(\boldsymbol{x}). \tag{8}$$

This yields the Newton update rule for second-order optimization:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma\left(\nabla^2\mathcal{L}(\boldsymbol{x}_t)\right)^{-1}\nabla\mathcal{L}(\boldsymbol{x}_t), \tag{9}$$

where $\gamma$ is a learning rate that controls the step size.

Compared to first-order methods such as gradient descent, second-order optimization offers several key advantages. First-order methods rely solely on gradient information and typically require many iterations to converge, especially when the loss surface exhibits high curvature, saddle points, or ill-conditioned regions. This slow convergence is particularly problematic in full-chip ILT, where the design space is extremely large and each lithography simulation is computationally expensive.

In contrast, second-order optimization leverages curvature information through the Hessian matrix to adaptively scale and orient the update direction. By incorporating this second-order information, the optimizer can make more informed updates, accelerating convergence by avoiding oscillations and inefficient zig-zagging in narrow valleys of the loss landscape. Consequently, second-order methods often reach high-quality solutions in significantly fewer iterations, making them desirable for computation-intensive tasks such as full-chip ILT.

Despite the advantages of second-order optimization, the high computation cost of constructing and inverting the full Hessian has traditionally hindered its practical use in large-scale ILT. For a scalar-valued loss function, the gradient vector $\nabla\mathcal{L}(\boldsymbol{x}) \in \mathbb{R}^{N^2}$ matches the dimensionality of the parameters $\boldsymbol{x}$. In contrast, the Hessian matrix $\nabla^2\mathcal{L}(\boldsymbol{x})$ lies in $\mathbb{R}^{N^2\times N^2}$, incurring memory usage that scales as $\mathcal{O}(N^4)$ and matrix inversion complexity of $\mathcal{O}(N^6)$. These prohibitive costs render exact Hessian computation impractical for large-scale problems like full-chip ILT, necessitating the adoption of efficient approximation techniques. Such approximations must be both memory- and computation-efficient to avoid the burdensome $\mathcal{O}(N^4)$ resource demands. Even widely used quasi-Newton methods like BFGS and L-BFGS [35], though more efficient, remain insufficient for problems of this scale due to memory and computation costs.

### B. Hessian Matrix Approximation

Fig. 3 presents a visual comparison between the gradient vector and the corresponding Hessian matrix at the first iteration during ILT optimization. The gradient vector captures the local slope of the loss function with respect to each mask parameter, while the Hessian matrix encodes second-order information, specifically the curvature of the loss landscape.

From the visualization, we observe that the Hessian matrix exhibits strong diagonal dominance—its diagonal elements are significantly larger in magnitude compared to the off-diagonal entries. This structure implies that the second derivative of the loss function with respect to each parameter is much more influential than the cross-derivatives between different parameters. Leveraging this insight, we can significantly reduce both memory and computation overhead by adopting a diagonal approximation of the Hessian. This not only enables scalable second-order optimization for full-chip ILT but also preserves much of the curvature information necessary for efficient convergence. Thus, diagonal dominance serves as both an empirical observation and a theoretical justification for simplifying the Hessian structure in large-scale mask optimization.

Based on the above discussion, the core idea of the proposed optimizer is to precondition the gradient using the inverse of Hessian matrix's diagonal, i.e., $diag\left(\nabla^2\mathcal{L}(\boldsymbol{x})\right)^{-1}$. The update step for the mask is then given by $diag\left(\nabla^2\mathcal{L}(\boldsymbol{x})\right)^{-1}\nabla\mathcal{L}(\boldsymbol{x}_t)$. This preconditioning effectively rescales the gradient to accelerate convergence.

This approach is particularly beneficial because the loss landscape in ILT is typically ill-conditioned—some regions exhibit high curvature (i.e., sharp) while others remain relatively flat. This characteristic is evident in the varying magnitudes of the diagonal elements of the Hessian matrix, as shown in Fig. 3. In such scenarios, applying a

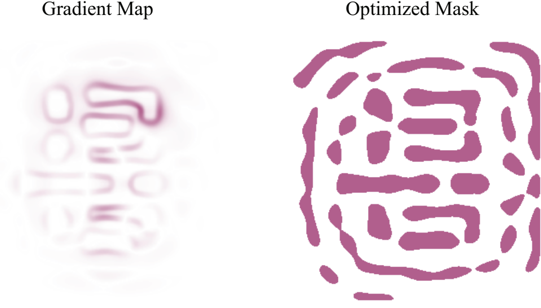


Fig. 4 Comparison between the gradient map and the optimized mask. The optimized mask includes main features in the center and SRAFs in the surrounding regions. However, standard gradient updates tend to emphasize the main patterns while underemphasizing the SRAFs, resulting in suboptimal ILT efficiency. This observation motivates the design of a gradient modulation strategy to better balance the optimization focus.

uniform step size in all directions can lead to suboptimal convergence. By contrast, the Hessian-based preconditioner adapts the update directions based on the local curvature, effectively stretching or contracting the directions as needed. Consequently, the optimizer can take larger steps in flatter regions and smaller steps in sharper regions, resulting in faster and more stable convergence.

Hutchinson's method [36], [37] provides an efficient way to approximate the diagonal of Hessian matrix. It is shown in [37] that:

$$diag\left(\nabla^2\mathcal{L}(\boldsymbol{x})\right) = \mathbb{E}\left[\boldsymbol{z}\odot\left(\nabla^2\mathcal{L}(\boldsymbol{x})\,\boldsymbol{z}\right)\right], \tag{10}$$

where $\boldsymbol{z}$ is a random vector following the Rademacher distribution—each entry independently takes the value $+1$ or $-1$ with equal probability. The product $\nabla^2\mathcal{L}(\boldsymbol{x})\,\boldsymbol{z}$ can be computed with:

$$\frac{\partial\left(\nabla\mathcal{L}(\boldsymbol{x})^T\boldsymbol{z}\right)}{\partial\boldsymbol{x}} = \frac{\partial\nabla\mathcal{L}(\boldsymbol{x})^T}{\partial\boldsymbol{x}}\boldsymbol{z} + \nabla\mathcal{L}(\boldsymbol{x})^T\frac{\partial\boldsymbol{z}}{\partial\boldsymbol{x}} = \nabla^2\mathcal{L}(\boldsymbol{x})\,\boldsymbol{z}, \tag{11}$$

since $\boldsymbol{z}$ is independent of $\boldsymbol{x}$ and thus its derivative vanishes. Equation (11) enables the computation of the Hessian-vector product required for the Hutchinson approximation without explicitly forming the full Hessian matrix, making the method highly efficient for large-scale optimization problems. Note that Hutchinson's method has also been adopted in the second-order optimizers for deep learning, such as AdaHessian [33] and Sophia [38].

### C. Exponential Moving Averages

Exponential Moving Average (EMA) is a widely used technique in optimization and training dynamics to smooth out parameter updates or loss curves over time. Unlike a simple moving average, which assigns equal weight to all data points within a fixed window, EMA assigns exponentially decreasing weights to older values. This gives more significance to recent data while still preserving the overall trend. The EMA of a variable $\boldsymbol{v}_t$ at step $t$ is computed as:

$$\overline{\boldsymbol{v}}_t = \beta\cdot\overline{\boldsymbol{v}}_{t-1} + (1-\beta)\cdot\boldsymbol{v}_t, \tag{12}$$

where $\beta\in(0,1)$ is a decay factor that controls the rate of forgetting. A larger $\beta$ results in smoother curves but slower adaptation to recent changes, while a smaller $\beta$ responds more quickly to recent updates. In the context of optimization, EMA is often used to stabilize training by averaging model weights or gradients across iterations, reducing the impact of noisy updates and improving generalization.

**Algorithm 1** Full-Chip ILT with Scalable Second-Order Optimizer

**Input:** Target image $\boldsymbol{Z}_Y$.
1: $\boldsymbol{x} = \boldsymbol{Z}_Y$;
2: **for** $t\in[1,2,...,\#Steps]$ **do** `//sequential execution;`
3: Partition the parameters into overlapping tiles;
4: Assign each tile $\boldsymbol{x}_t^{(i)}$ to a GPU;
5: **for** $i\in[1,2,...,\#Tiles]$ **do** `//parallel execution;`
6: Generate printed images via Equations (1), (2), and (3);
7: Calculate the loss function $\mathcal{L} = \mathcal{L}_{L2} + \mathcal{L}_{PVB}$ based on Equations (4) and (5);
8: Update the EMA of gradient via Equation (13);
9: **if** $t\%S_H = 0$ **then**
10: Update the EMA of Hessian via Equation (14);
11: **end if**
12: Compute the mask update for $\boldsymbol{x}_t^{(i)}$ using Equation (16);
13: **end for**
14: Synchronize to update the global parameters;
15: **end for**
16: **return** $\boldsymbol{x}$;

In this paper, we employ EMA to the gradients and Hessian matrices. As introduced in [32], initialization bias correction should be applied to the EMA result since it is initialized as all zeros. Therefore, the gradient used in mask update is given by:

$$\overline{\nabla\mathcal{L}(\boldsymbol{x}_t)} = \frac{\beta_1\cdot\overline{\nabla\mathcal{L}(\boldsymbol{x}_{t-1})} + (1-\beta_1)\cdot\nabla\mathcal{L}(\boldsymbol{x}_t)}{1-\beta_1^t}. \tag{13}$$

Similarly, the Hessian matrix used in mask update is given by:

$$\overline{\nabla^2\mathcal{L}(\boldsymbol{x}_t)} = \frac{\beta_2\cdot\overline{\nabla^2\mathcal{L}(\boldsymbol{x}_{t-1})} + (1-\beta_2)\cdot diag\left(\nabla^2\mathcal{L}(\boldsymbol{x}_t)\right)}{1-\beta_2^t}. \tag{14}$$

With our EMA strategy, the mask update rule at each step becomes:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma\left(\overline{\nabla^2\mathcal{L}(\boldsymbol{x}_t)}\right)^{-1}\overline{\nabla\mathcal{L}(\boldsymbol{x}_t)}. \tag{15}$$

Note that we adopt the Hutchinson approximation given by Equation (10) and Equation (11) to compute $diag\left(\nabla^2\mathcal{L}(\boldsymbol{x}_t)\right)$. This enables efficient mask update based on Equation (16). To further reduce the computation, we can update the EMA of Hessian matrix per $S_H \gg 1$ steps, making the cost of Hutchinson approximation almost negligible. In this paper, we use $S_H = 16$.

### D. Gradient Modulation

Fig. 4 illustrates a comparison between the gradient map and the final optimized mask, highlighting an important challenge in ILT. In the optimized mask, the central region contains the main patterns that correspond directly to the intended device features. Surrounding these main patterns are SRAFs, which are critical for enhancing image fidelity during lithography, especially under aggressive resolution constraints. Despite the importance of both pattern types, the gradient map shows that conventional gradient descent tends to prioritize regions with larger intensity gradients, typically the main patterns. As a result, SRAFs, which often exhibit smaller gradients due to their subtle influence on the aerial image, receive less attention during optimization. This imbalance leads to slower convergence or suboptimal SRAFs, potentially compromising printability and pattern fidelity under varying process conditions.

This discrepancy highlights the inefficiency of using a uniform gradient update strategy in ILT, where both main features and SRAFs serve distinct yet equally important roles. To address this issue, we

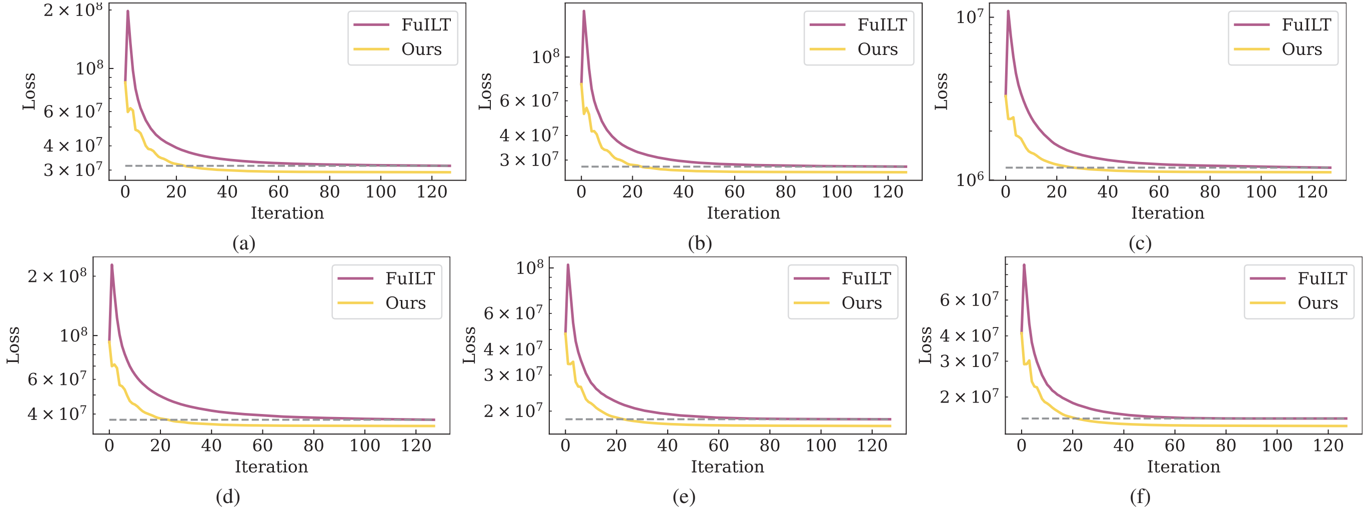


Fig. 5 Comparison of loss curves between FuILT and our method on the (a) `aes`, (b) `dyn_node`, (c) `gcd`, (d) `ibex`, (e) `pico`, (f) `riscv32i` testcases. Across all testcases, our approach outperforms FuILT within just 24 iterations. Our second-order optimizer enables faster convergence and lower loss values compared to FuILT.

propose a gradient modulation strategy aimed at rebalancing the optimization effort across the mask. Under this strategy, the mask update rule at each ILT iteration is given by:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \, sign\left( \left( \overline{\nabla^2 \mathcal{L}(\boldsymbol{x}_t)} \right)^{-1} \overline{\nabla \mathcal{L}(\boldsymbol{x}_t)} \right), \quad (16)$$

where $sign(\cdot)$ is applied element-wise and maps positive values to $+1$ and negative values to $-1$. This modulation selectively enhances the influence of gradients in regions corresponding to SRAFs, ensuring that these features receive sufficient attention during optimization. By better balancing the updates between main features and SRAFs, this approach improves pattern fidelity with fewer iterations.

### *E. Summary*

In summary, we propose a scalable second-order optimizer to improve the convergence efficiency and solution quality of full-chip ILT. As shown in Algorithm 1, the optimization process begins by initializing the mask parameters $\boldsymbol{x}$ with the target pattern $\boldsymbol{Z}_Y$ (line 1). At each iteration, the full-chip parameters are partitioned into overlapping tiles, which are distributed across multiple GPUs to enable parallel processing (lines 3-4). For each tile, lithography simulation is conducted to generate printed patterns (line 6). The loss function is then computed as a combination of the L2 loss and PVB loss, capturing both fidelity to the target pattern and robustness to process variation (line 7). In the mask update step, our second-order optimization strategy is applied. This update incorporates Hessian matrix approximation, EMA, and gradient modulation to achieve efficient and balanced optimization (lines 8-12). Once the updates are computed for all tiles in parallel, they are synchronized globally to ensure consistency across overlapping regions (line 14).

## IV. Experiments

We assess the effectiveness of full-chip ILT algorithms using six large-scale benchmarks: `aes`, `dyn_node`, `gcd`, `ibex`, `pico`, and `riscv32i`. These layouts are generated via the OpenROAD design flow [43], utilizing the NanGate 45 nm process design kit (PDK) [44]. For each design, we extract the metal-1 layer as the optimization target, since it typically features dense, long-range patterns and poses the greatest challenge in full-chip ILT. TABLE I summarizes our full-chip ILT benchmark. In contrast to the small $2 \times 2\,\mu\text{m}^2$ layouts commonly used in prior ILT studies [2], [45], our benchmarks are substantially larger and present a more demanding testbed for ILT.

TABLE I Information about our full-chip ILT benchmark.

| Testcase | Layout Size | Description |
|---|---|---|
| `aes` | $240 \times 240\,\mu\text{m}^2$ | AES encryption circuit [39] |
| `dyn_node` | $240 \times 240\,\mu\text{m}^2$ | 2D mesh crossbar [40] |
| `gcd` | $32 \times 32\,\mu\text{m}^2$ | GCD computation circuit [39] |
| `ibex` | $240 \times 240\,\mu\text{m}^2$ | Ibex RISC-V core [41] |
| `pico` | $184 \times 184\,\mu\text{m}^2$ | Size-optimized RISC-V CPU [42] |
| `riscv32i` | $170 \times 170\,\mu\text{m}^2$ | Simple RISC-V 32I CPU [39] |

All experiments are conducted on NVIDIA RTX 3090 GPUs. The ILT algorithms are implemented based on PyTorch [46] and OpenILT [47], with the optical kernels provided by the ICCAD-13 contest [48]. Each layout is converted into a grayscale image, where each pixel corresponds to an $8 \times 8\,\text{nm}^2$ region of the physical design. FuILT [31] serves as the baseline in our experiments, and our method builds upon it by incorporating the proposed second-order optimizer. By default, all ILT algorithms are run for 128 iterations. FuILT employs a gradient descent optimizer[1] with a learning rate of $0.5$, while our solver uses a learning rate of $0.1$. These values are selected as the optimal settings based on a grid search.

Fig. 5 presents the comparison of loss curves, clearly demonstrating the efficiency and effectiveness of the proposed second-order optimizer. Across all testcases, our method consistently outperforms FuILT, achieving lower loss values in significantly fewer iterations. Notably, our optimizer surpasses FuILT within just 24 iterations, highlighting its ability to quickly refine the mask and approach high-quality solutions early in the optimization process.

TABLE II compares the performance of our method with FuILT. In addition to the full configuration with 128 iterations, we also report the results of a fast configuration using only 32 iterations. Our full configuration achieves a $22.5\%$ improvement in L2 and a $36.9\%$

[1] In our preliminary experiments, the Adam optimizer, while often more effective in deep learning, performed significantly worse than vanilla gradient descent. Therefore, we use gradient descent as the optimizer for FuILT.

TABLE II Comparison of full-chip ILT methods.

| Testcase | Layout Size | FuILT [31] | | | | Ours-Fast | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | L2† | PVB† | EPE | #Steps | L2† | PVB† | EPE | #Steps | L2† | PVB† | EPE | #Steps |
| aes | $240 \times 240\,\mu m^2$ | 1123.0 | 1918.8 | 755.3k | 128 | 1087.6 | 1918.0 | 688.9k | 32 | 875.6 | 1943.9 | 518.1k | 128 |
| dyn_node | $240 \times 240\,\mu m^2$ | 993.5 | 1747.2 | 1000.7k | 128 | 933.0 | 1743.6 | 603.0k | 32 | 769.7 | 1772.2 | 490.7k | 128 |
| gcd | $32 \times 32\,\mu m^2$ | 23.1 | 40.3 | 10.7k | 128 | 22.3 | 40.2 | 10.0k | 32 | 17.8 | 40.7 | 7.6k | 128 |
| ibex | $240 \times 240\,\mu m^2$ | 1402.7 | 2291.7 | 805.5k | 128 | 1311.3 | 2299.4 | 733.6k | 32 | 1082.7 | 2336.5 | 569.4k | 128 |
| pico | $184 \times 184\,\mu m^2$ | 660.3 | 1115.1 | 403.9k | 128 | 632.5 | 1113.4 | 369.2k | 32 | 522.1 | 1122.8 | 286.8k | 128 |
| riscv32i | $170 \times 170\,\mu m^2$ | 585.4 | 960.5 | 381.5k | 128 | 544.6 | 959.8 | 321.6k | 32 | 445.7 | 966.5 | 248.3k | 128 |
| Average | - | 798.0 | **1345.6** | 559.6k | 128 | 755.2 | 1345.7 | 454.3k | **32** | **618.9** | 1363.7 | **353.4k** | 128 |
| Ratio | - | 100.0% | **100.0%** | 100.0% | 100.0% | 94.6% | 100.0% | 81.1% | **25.0%** | **77.5%** | 101.3% | **63.1%** | 100.0% |
| Score‡ | - | $7.9\times10^9$ (100.0%) | | | | $7.6\times10^9$ (96.2%) | | | | **$7.2\times10^9$ (91.4%)** | | | |

† The unit of L2 and PVB is $\mu m^2$.
‡ Score = $4 \cdot$PVB + $5000 \cdot$EPE.

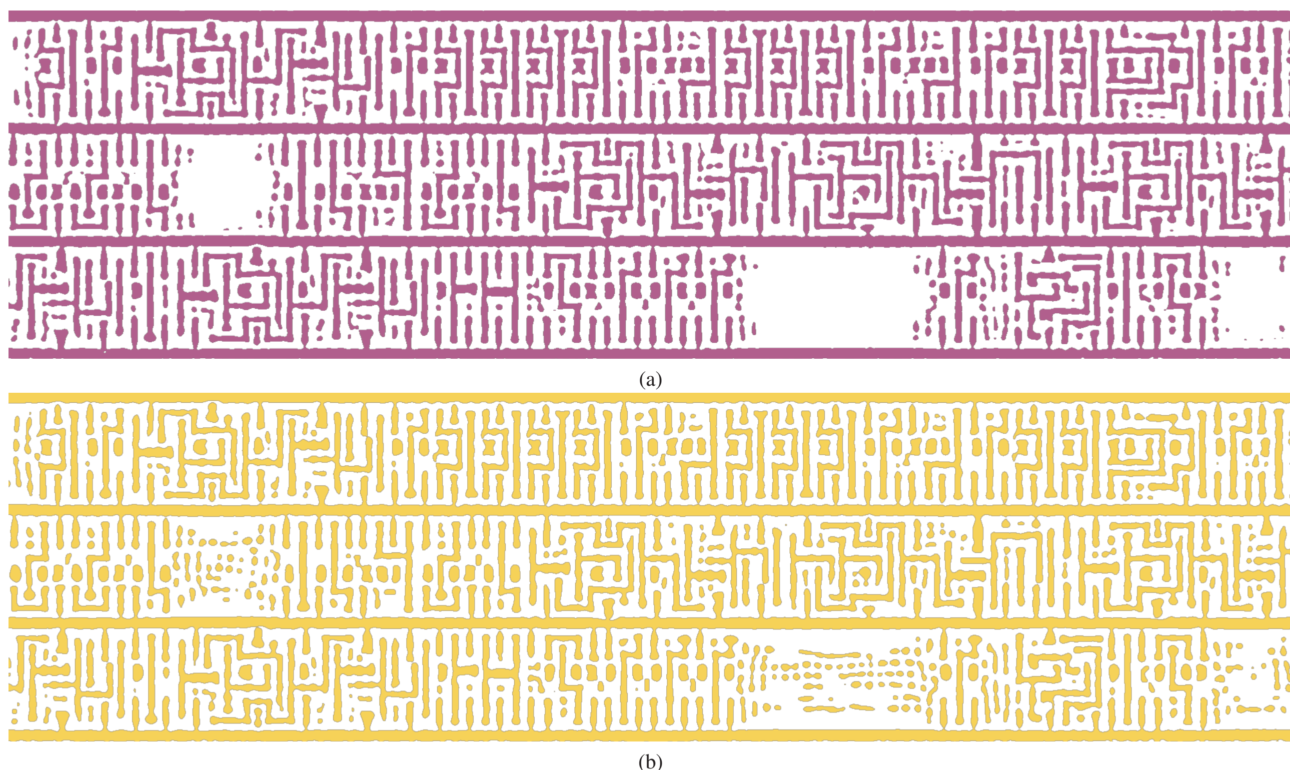

Fig. 6 Examples of full-chip ILT results from (a) FuILT and (b) our method. The mask crops are taken from a small region of the *gcd* testcase. Our method can generate better SRAFs to improve the mask quality.

reduction in EPE compared to FuILT. In terms of the composite score, defined as $4 \cdot \text{PVB} + 5000 \cdot \text{EPE}$, our method outperforms FuILT by $8.6\%$. Even in the fast configuration, our method achieves better L2 and EPE than FuILT while requiring $75.0\%$ fewer iterations. Importantly, each iteration in our method takes a similar amount of time as FuILT. The Hessian update, which is approximately twice as costly as a gradient update, is only performed once every 16 iterations, ensuring minimal overhead. Fig. 6 presents examples of full-chip ILT results from FuILT and our method.

Empirical results demonstrate that our approach not only reduces the number of required iterations but also consistently delivers superior mask quality across a wide range of full-chip designs. These improvements translate to shorter optimization runtimes and enhanced pattern fidelity, effectively addressing key bottlenecks in conventional ILT pipelines. Overall, the proposed method offers a practical, scalable solution for advanced ILT tasks, making it highly suitable for real-world deployment.

## V. Conclusion

We present a scalable second-order optimizer tailored for full-chip ILT. By leveraging the diagonal of Hessian matrix through Hutchinson's method, our approach incorporates curvature information with minimal computational and memory overhead. We further introduce EMA and gradient modulation to enhance convergence stability and mask quality. Extensive experiments on industry-scale layouts demonstrate that our method significantly accelerates convergence and improves pattern fidelity compared to the first-order baseline. Our optimizer reduces iteration counts while maintaining runtime efficiency, offering a practical and effective solution for large-scale ILT challenges in advanced semiconductor manufacturing.

# 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," *Integration, the VLSI Journal*, vol. 77, pp. 96–103, 2021.

[2] S. Zheng, H. Yang, B. Zhu, B. Yu, and M. Wong, "LithoBench: Benchmarking ai computational lithography for semiconductor manufacturing," in *Proc. NeurIPS*, 2023.

[3] J. Ou, B. Yu, J.-R. Gao, D. Z. Pan, M. Preil, and A. Latypov, "Directed self-assembly based cut mask optimization for unidirectional design," in *Proc. GLSVLSI*, 2015, pp. 83–86.

[4] Y. Ma, X. Zeng, and B. Yu, "Methodologies for layout decomposition and mask optimization: A systematic review," in *Proc. VLSI-SoC*, 2017.

[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. ICCAD*, 2017, pp. 81–88.

[6] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," in *Proc. DAC*, 2020.

[7] X. Liang, Y. Ouyang, H. Yang, B. Yu, and Y. Ma, "RL-OPC: Mask optimization with deep reinforcement learning," in *IEEE TCAD*, vol. 43, no. 1, 2024, pp. 340–351.

[8] X. Liang, H. Yang, K. Liu, B. Yu, and Y. Ma, "CAMO: Correlation-aware mask optimization with modulated reinforcement learning," in *Proc. DAC*, 2024.

[9] S. Zheng, Y. Ma, B. Yu, and M. Wong, "EMOGen: Enhancing mask optimization via pattern generation," in *Proc. DAC*, 2024.

[10] W. Zhao, X. Yao, S. Yin, Y. Bai, Z. Yu, Y. Ma, B. Yu, and M. D. Wong, "AdaOPC 2.0: Enhanced adaptive mask optimization framework for via layers," *IEEE TCAD*, 2024.

[11] G. Chen, H. Yang, H. Ren, B. Yu, and D. Z. Pan, "Differentiable edge-based OPC," in *Proc. ICCAD*, 2024.

[12] J.-S. Park, C.-H. Park, S.-U. Rhie, Y.-H. Kim, M.-H. Yoo, J.-T. Kong, H.-W. Kim, and S.-I. Yoo, "An efficient rule-based opc approach using a drc tool for 0.18/spl mu/m asic," in *Proc. ISQED*, 2000, pp. 81–85.

[13] S. O'Brien, R. Soper, S. Best, and M. Mason, "Rules based process window OPC," in *Design for Manufacturability through Design-Process Integration II*, vol. 6925, 2008, pp. 396–404.

[14] J. Wang, A. Wei, P. Verma, and W. Wilkinson, "Optimization of rule-based OPC fragmentation to improve wafer image rippling," in *European Mask and Lithography Conference*, vol. 9661, 2015, pp. 79–94.

[15] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama, "A fast process variation and pattern fidelity aware mask optimization algorithm," in *Proc. ICCAD*, 2014, pp. 238–245.

[16] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *Proc. DATE*, 2015, pp. 1591–1594.

[17] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," *IEEE TCAD*, vol. 35, no. 8, pp. 1345–1357, 2016.

[18] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *Proc. DATE*, 2015, pp. 1591–1594.

[19] Z. Yu, S. Zheng, W. Zhao, S. Yin, X. Liang, G. Chen, Y. Ma, B. Yu, and M. D. Wong, "RuleLearner: OPC rule extraction from inverse lithography technique engine," *IEEE TCAD*, 2024.

[20] S. Zheng, G. Xiao, G. Yan, M. Dong, Y. Li, H. Chen, Y. Ma, B. Yu, and M. Wong, "Model-based OPC extension in OpenILT," in *Proc. ISEDA*, 2024, pp. 568–573.

[21] S. Zheng, X. Liang, Z. Yu, Y. Ma, B. Yu, and M. Wong, "Curvilinear optical proximity correction via cardinal spline," in *Proc. DAC*, 2025.

[22] Y. Liu, D. Abrams, L. Pang, and A. Moore, "Inverse lithography technology principles in practice: Unintuitive patterns," in *BACUS Symposium on Photomask Technology*, vol. 5992, 2005, pp. 886–893.

[23] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. DAC*, 2014.

[24] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level set method for mask optimization," in *Proc. DATE*, 2021, pp. 1835–1838.

[25] S. Sun, F. Yang, B. Yu, L. Shang, and X. Zeng, "Efficient ilt via multi-level lithography simulation," in *Proc. DAC*, 2023.

[26] B. Zhu, S. Zheng, Z. Yu, G. Chen, Y. Ma, F. Yang, B. Yu, and M. D. Wong, "L2O-ILT: Learning to optimize inverse lithography techniques," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 3, pp. 944–955, 2023.

[27] S. Sun, F. Yang, B. Yu, L. Shang, D. Zhou, and X. Zeng, "Efficient ILT via Multigrid-Schwartz method," in *Proc. DAC*, 2024.

[28] Y. Luo, X. Liang, and Y. Ma, "Enabling robust inverse lithography with rigorous multi-objective optimization," in *Proc. ICCAD*, 2024.

[29] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020.

[30] L. Pang, "Inverse lithography technology: 30 years from concept to practical, full-chip reality," *Journal of Micro/Nanopatterning, Materials, and Metrology*, vol. 20, no. 3, 2021.

[31] S. Yin, W. Zhao, L. Xie, H. Chen, Y. Ma, T.-Y. Ho, and B. Yu, "FuILT: Full chip ILT system with boundary healing," *Proc. ISPD*, 2024.

[32] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "AdaHessian: An adaptive second order optimizer for machine learning," in *Proc. AAAI*, vol. 35, no. 12, 2021, pp. 10 665–10 673.

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

[35] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[36] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines," *Communications in Statistics-Simulation and Computation*, vol. 18, no. 3, pp. 1059–1076, 1989.

[37] C. Bekas, E. Kokiopoulou, and Y. Saad, "An estimator for the diagonal of a matrix," *Applied numerical mathematics*, vol. 57, no. 11-12, pp. 1214–1229, 2007.

[38] H. Liu, Z. Li, D. Hall, P. Liang, and T. Ma, "Sophia: A scalable stochastic second-order optimizer for language model pre-training," 2023.

[39] O. Contributors, "Openroad flow," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts, 2020.

[40] G. Tziantzioulis, T.-J. Chang, J. Balkind, J. Tu, F. Gao, and D. Wentzlaff, "OPDB: A scalable and modular design benchmark," *IEEE TCAD*, vol. 41, no. 6, pp. 1878–1887, 2021.

[41] I. Contributors, "Ibex risc-v core," https://github.com/lowRISC/ibex, 2017.

[42] P. Contributors, "Picorv32 - a size-optimized risc-v cpu," https://github.com/YosysHQ/picorv32, 2019.

[43] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proc. DAC*, 2019.

[44] "Nangate 45nm library," http://www.nangate.com/.

[45] B. Jiang, L. Liu, Y. Ma, B. Yu, and E. F. Young, "Neural-ILT 2.0: Migrating ilt to domain-specific and multitask-enabled neural network," *IEEE TCAD*, vol. 41, no. 8, pp. 2671–2684, 2021.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[47] S. Zheng, B. Yu, and M. Wong, "OpenILT: An open source inverse lithography technique framework," in *Proc. ASICON*, 2023.

[48] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. ICCAD*, 2013, pp. 271–274.