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

Bentian Jiang[1], Lixin Liu[1], Yuzhe Ma[1], Hang Zhang[2], Bei Yu[1] and Evangeline F.Y. Young[1]

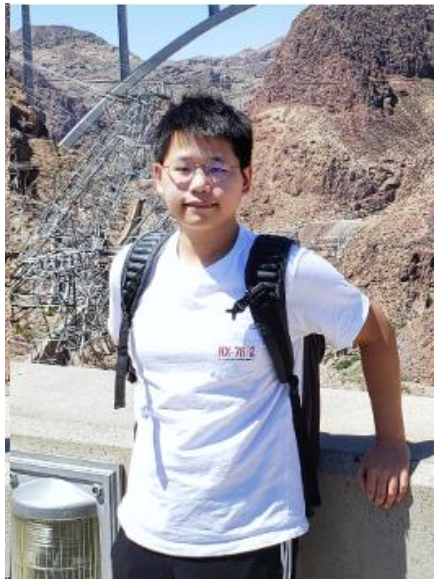
[1] CSE Dept., The Chinese University of Hong Kong
[2]ECE Dept., Cornell University

Speaker Biography

- Bentian Jiang is currently pursuing a Ph.D. degree with the Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, under the supervision of Prof. Evangeline F.Y. Young.

- He is a recipient of several prizes in renowned EDA contests including the CAD Contests at ICCAD 2018 and ISPD 2018, 2019, 2020.

- His research interests include
  - Design for manufacturability
  - Physical design



Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion

Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion

Background

Lithography

- Use light to transfer a geometric pattern from a photomask to a light-sensitive photoresist on the wafer
- Mismatch between lithography system and device feature sizes

Optical proximity correction (OPC)

- OPC compensates the printing errors by modifying the mask layouts
- Compact lithography simulation model (designed to learn the printing effects) can guide the model-based OPC processes

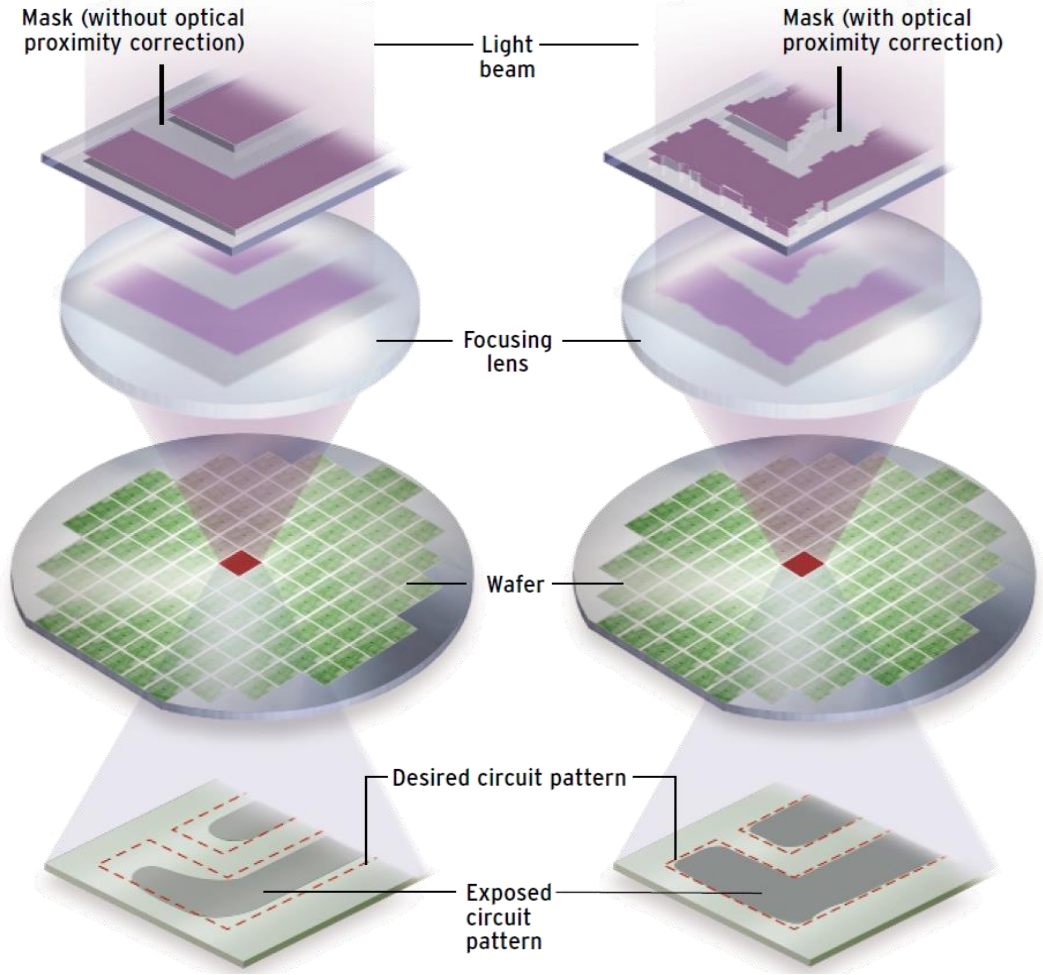


Figure sources from F. Schellenberg†

† F. Schellenberg, "A little light magic [optical lithography]," in IEEE Spectrum, vol. 40, no. 9, pp. 34-39, Sept. 2003, doi: 10.1109/MSPEC.2003.1228007.

Inverse Lithography Technology (ILT)

- Forward lithography simulation can mimic the mask printing effects on wafer
  - Given the desired target pattern $\mathbf{Z}_t$, optimized mask $\mathbf{M}$
  - Forward Lithography simulation produce the corresponding wafer image

$$\mathbf{Z} = f(\mathbf{M}\,;\mathbf{P}_{\mathrm{nom}})$$

- ILT correction tries to find the optimum mask $\mathbf{M}_{\mathrm{opt}}$

$$\mathbf{M}_{\mathrm{opt}} = f^{-1}(\mathbf{Z}_{\mathrm{t}};\mathbf{P}_{\mathrm{nom}})$$

- Features
  - **Ill-posed**: no explicit closed-form solution for $f^{-1}(\cdot\,;\mathbf{P}_{\mathrm{nom}})$
  - **Numerical**: gradient descent to update the on-mask pixels iteratively
  - Pros: best possible overall process window [1] [2] for 193i layers and EUV
  - Cons: drastically computational overhead, unmanageable mask writing time

Motivations

- Tremendous demands
  - Quality: best possible process window obtainable for 193i and EUV layers [1] [2]
  - Manufacturability: unmanageable mask writing times of ideal ILT curvilinear shapes affect high-volume yields
  - Affordability: the still increasing computational overhead

- Goals
  - A purely learning-based end-to-end ILT solution
    - The satisfactory mask printing shapes
    - Breakthrough reduction on computational overhead
    - Significant improvement on mask shape complexity
    - …
  - A learning-scheme with performance guarantee

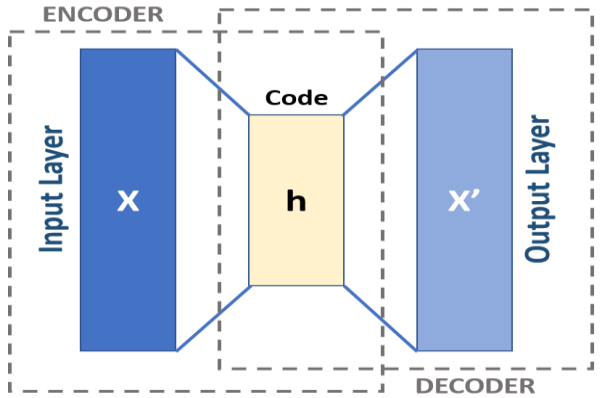
Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion

Why Neural Network – Analogy

- What kind of container is need for end-to-end ILT correction process
  - Layout image in, mask image out
  - Iterative process
  - Update an “object” (mask here) iteratively by gradient descent

- Does it sound like the **training procedure** of an auto-encoder network?
  - Encoder + decoder -> Image in, image out
  - Iteratively update neurons of each layer by gradient descent



Schema of a basic Autoencoder
By Michela Massi - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=80177333

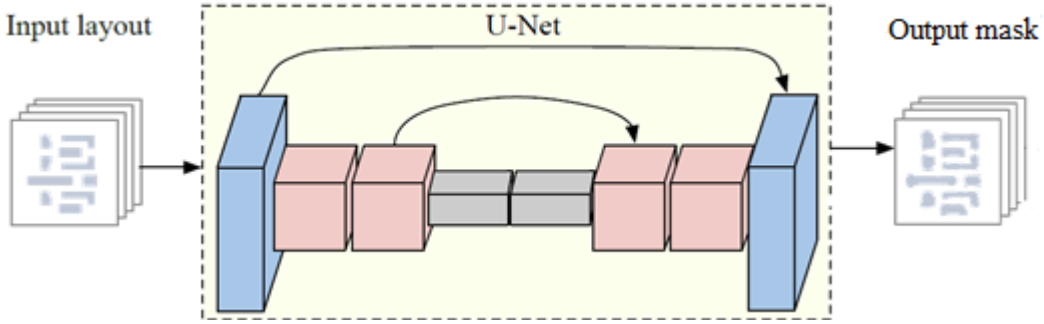
Starting from Scratch

- Let us start Neural-ILT with a basic image-to-image translation task

- Given the sets of
  - Input target layouts $\mathcal{Z}_t = \{\mathbf{Z}_{t,1}, \mathbf{Z}_{t,2}, \mathbf{Z}_{t,3}, \dots, \mathbf{Z}_{t,n}\}$
  - Corresponding ILT synthesized mask set $\mathcal{M}^* = \{\mathbf{M}_1^*, \mathbf{M}_2^*, \mathbf{M}_3^*, \dots, \mathbf{M}_n^*\}$

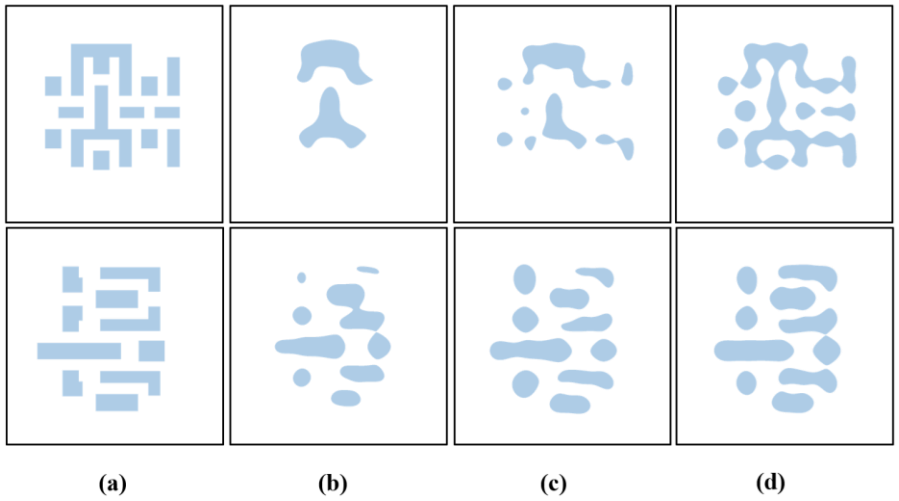
- The training procedure (supervised) of the UNet is to minimize the objective:

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



Untrustworthy Quality of Prediction

- Big trouble – Untrustworthy predict quality



(a) Target layouts.

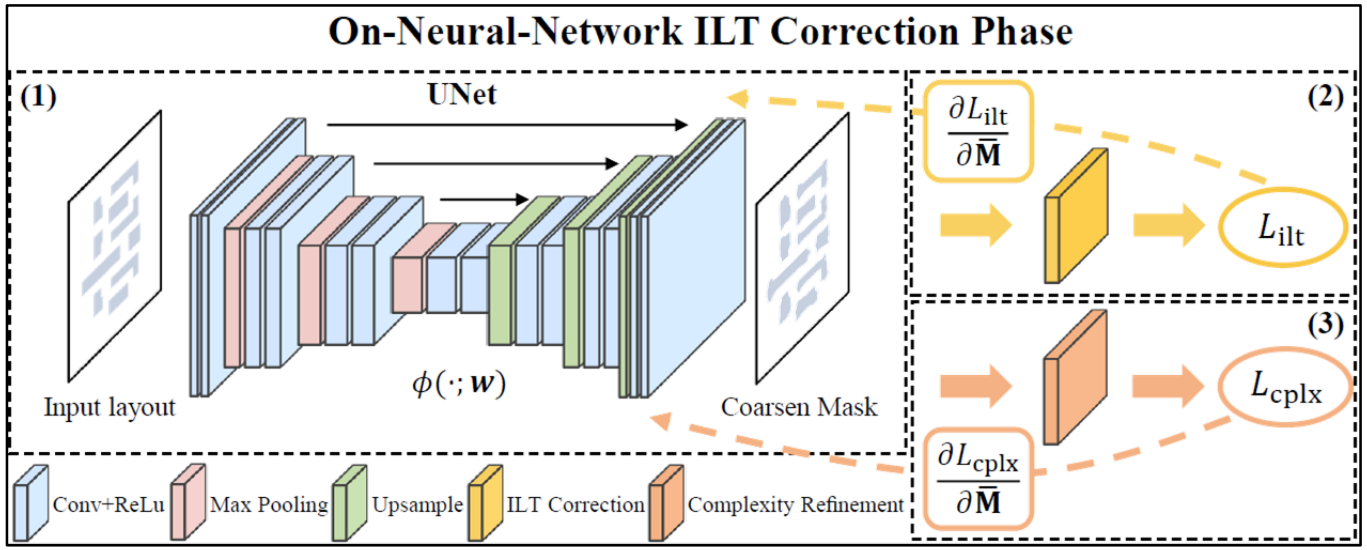
Wafer images generated by:
(b) Target layouts
(c) UNet direct prediction
(d) ILT synthesized masks

- Exists inevitable prediction loss which is not acceptable

- On-neural-network ILT correction is needed to ensure performance
  - Our solution: cast ILT as an **unsupervised** neural-network training procedure

Overview of Neural-ILT

- 3 sub-units:
  - A pre-trained UNet for performing layout-to-mask translation
  - An ILT correction layer for minimizing inverse lithography loss
  - A mask complexity refinement layer for removing redundant complex features
- Core engine:
  - CUDA-based lithography simulator (a partially coherent imaging model)



Challenges on Runtime Bottleneck

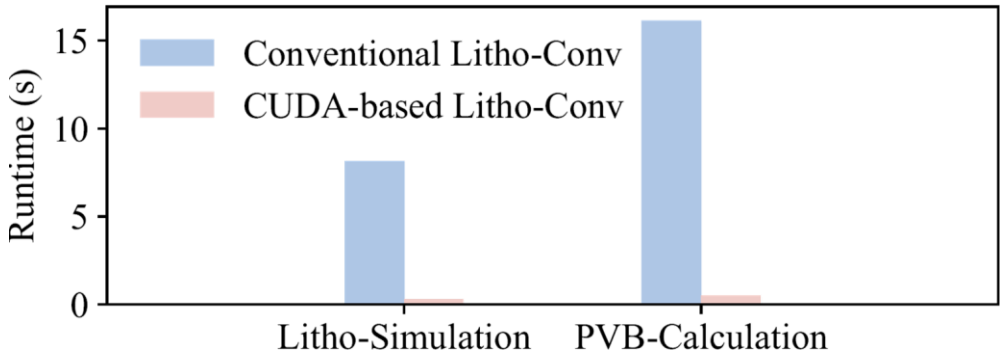
- Main computational overhead of ILT correction lies in mask litho-simulation
- Multiple rounds of litho-simulation (per layout, per iteration) are indispensable for guiding the ILT correction
- First critical challenge is to integrate a **fast-enough** lithography simulator into our Neural-ILT framework

GPU-based Litho-Simulator

- Partially coherent imaging system for lithography model $f(\mathbf{M}; \mathbf{P}_{\text{nom}})$
  - Given the mask **M**, litho-sim model parameters $\omega_k, \boldsymbol{h}_k$, wafer image **Z** can be calculated as

$$\mathbf{I}(x,y) = \sum_{k=1}^{N^2} \omega_k |\mathbf{M}(x,y) \otimes \mathbf{h}_k(x,y)|^2 \qquad \mathbf{Z}(x,y) = \begin{cases} 1, & \text{if } \mathbf{I}(x,y) \geq I_{th} \\ 0, & \text{if } \mathbf{I}(x,y) < I_{th} \end{cases}$$

- CUDA: perfect for parallelization + demands of AI toolkits integration
  - **96%** reduction in litho-simulation time
  - **97%** reduction in PVBand calculation time
  - Compatible with popular toolkits: PyTorch, TensorFLow, etc…



ILT Correction Layer

- ILT correction is essentially minimizing the images difference by gradient descent

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

- Gradient of L_{ilt} with respect to mask $\bar{\mathbf{M}}$ ($\mathbf{M} = \text{sigmoid}(\bar{\mathbf{M}})$) can be derived as

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

  - where $\mathbf{Z}_{\text{t}}$ is target pattern, $\mathbf{Z}$ is wafer image, $\mathbf{M}$ is mask, $\omega_k, \boldsymbol{h}_k$ are litho-sim model parameters

ILT Correction Layer

- ILT Correction Layer Implementation
  - Forward to calculate the ilt loss with respect to network prediction and target layout
  - Backward to calculate the gradient mask to update the UNet neurons
- Extremely fast with our GPU-based lithography simulator
- Directly used as a successor layer of other neural networks (expressed in PyTorch)

Algorithm 2 ILT Correction Layer Forward and Backward

Input: Masks $\mathbf{M}$, $\bar{\mathbf{M}}$, target layout $\mathbf{Z}_t$, kernels $\mathbf{H}$, $\mathbf{H}^*$, weights ω.

Forward

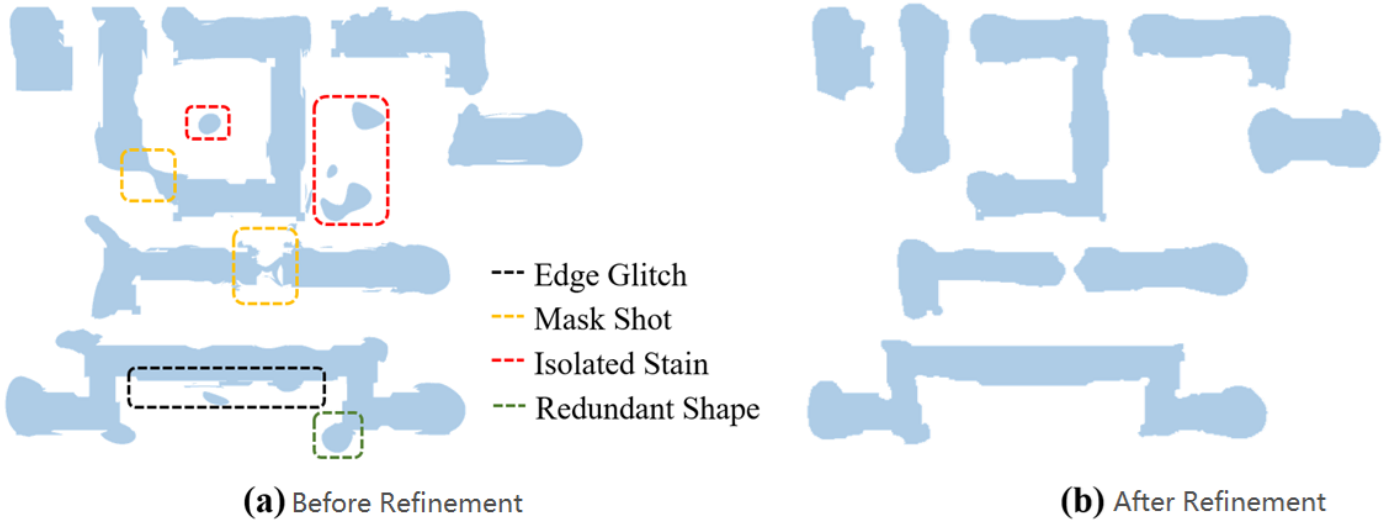
1: **function** `Forward`($\mathbf{M}$, $\mathbf{H}$, ω)
2: $\quad\mathbf{I}, \mathbf{Z} \leftarrow$ `CUDA_LITHO`($\mathbf{M}$, $\mathbf{H}$, ω, 1.0, SIMULATION);
3: $\quad L_{\text{ilt}} \leftarrow ||\mathbf{Z} - \mathbf{Z}_t||_\gamma^\gamma$; ▷ $\gamma = 4$ in forward
4: $\quad$**return** Lithography loss L_{ilt};

Backward

5: **function** `Backward`($\mathbf{M}$, $\bar{\mathbf{M}}$, $\mathbf{H}$, $\mathbf{H}^*$, ω) ▷ $\theta_M = 4$, $\theta_Z = 50$
6: $\quad\mathbf{I}, \mathbf{Z} \leftarrow$ `CUDA_LITHO`($\mathbf{M}$, $\mathbf{H}$, ω, 1.0, SIMULATION);
7: $\quad\mathbf{Z} \leftarrow \frac{1}{1+\exp(-\theta_Z \times (\mathbf{I} - I_{th}))}$, $\mathbf{M} \leftarrow \frac{1}{1+\exp(-\theta_M \times \bar{\mathbf{M}})}$;
8: $\quad$Define common term as $\mathbf{T}_C$, gradient left term as $\mathbf{G}_L$, gradient right term as $\mathbf{G}_R$;
9: $\quad\mathbf{T}_C \leftarrow (\mathbf{Z} - \mathbf{Z}_t)^{\gamma-1} \odot \mathbf{Z} \odot (1 - \mathbf{Z})$;
10: $\quad\mathbf{G}_L \leftarrow \mathbf{T}_C \odot$ `CUDA_LITHO`($\mathbf{M}$, $\mathbf{H}^*$, ω, 1.0, CONVOLVE);
11: $\quad\mathbf{G}_R \leftarrow \mathbf{T}_C \odot$ `CUDA_LITHO`($\mathbf{M}$, $\mathbf{H}$, ω, 1.0, CONVOLVE);
12: $\quad\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}} \leftarrow \gamma\theta_M\theta_Z \times$ [`CUDA_LITHO`($\mathbf{G}_L$, $\mathbf{H}^{\text{flip}}$, ω, 1.0, CONVOLVE) + `CUDA_LITHO`($\mathbf{G}_R$, $(\mathbf{H}^{\text{flip}})^*$, ω, 1.0, CONVOLVE)] $\odot \mathbf{M} \odot (1 - \mathbf{M})$; ▷ Compute Equation (5) using Algorithm 1
13: $\quad$**return** Gradient $\frac{\partial L_{\text{ilt}}}{\partial \bar{\mathbf{M}}}$;

Complexity Refinement Layer

- ILT synthesized masks
  - Non-rectangular complex shapes
  - Not manufacturing-friendly
- Complex features
  - Isolated curvilinear stains
  - Edge glitches
  - Redundant contours
- Goals
  - Eliminate the redundant/complex features
  - Maintain competitive mask printability



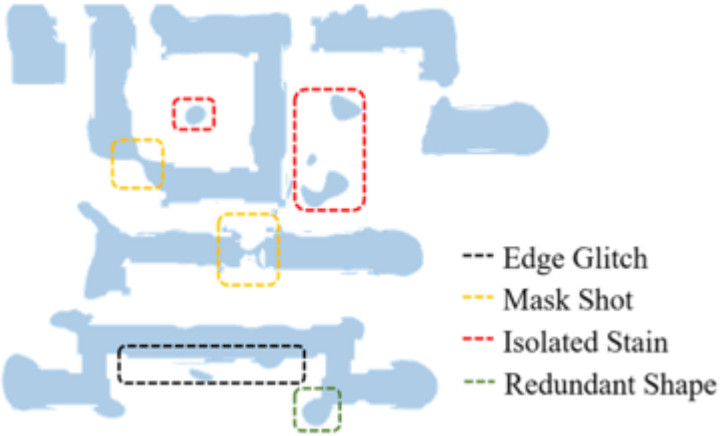
Complexity Refinement Layer

- Complex features are distributed around/on the original patterns

- Observe that, those features
  - Help to improve printability under nominal process condition
  - Not printed under min ($\mathbf{P}_{\text{min}}$) / nominal ($\mathbf{P}_{\text{nom}}$) process conditions
  - But usually printed under max process condition ($\mathbf{P}_{\text{max}}$)

- Cause area variations between
  - $\mathbf{Z}_{\text{in}} = f(\mathbf{M}; \mathbf{P}_{\text{min}})$ and $\mathbf{Z}_{\text{out}} = f(\mathbf{M}; \mathbf{P}_{\text{max}})$
  - Loss function: $L_{\text{cplx}} = ||\mathbf{Z}_{\text{in}} - \mathbf{Z}_{\text{out}}||_2^2$.

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

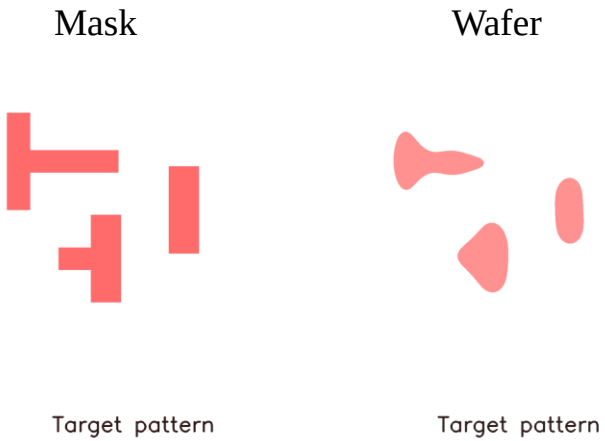


Neural-ILT

- 3 sub-units:
  - A pre-trained UNet for performing layout-to-mask translation
  - An ILT correction layer for minimizing lithography loss
  - A mask complexity refinement layer for removing redundant complex features

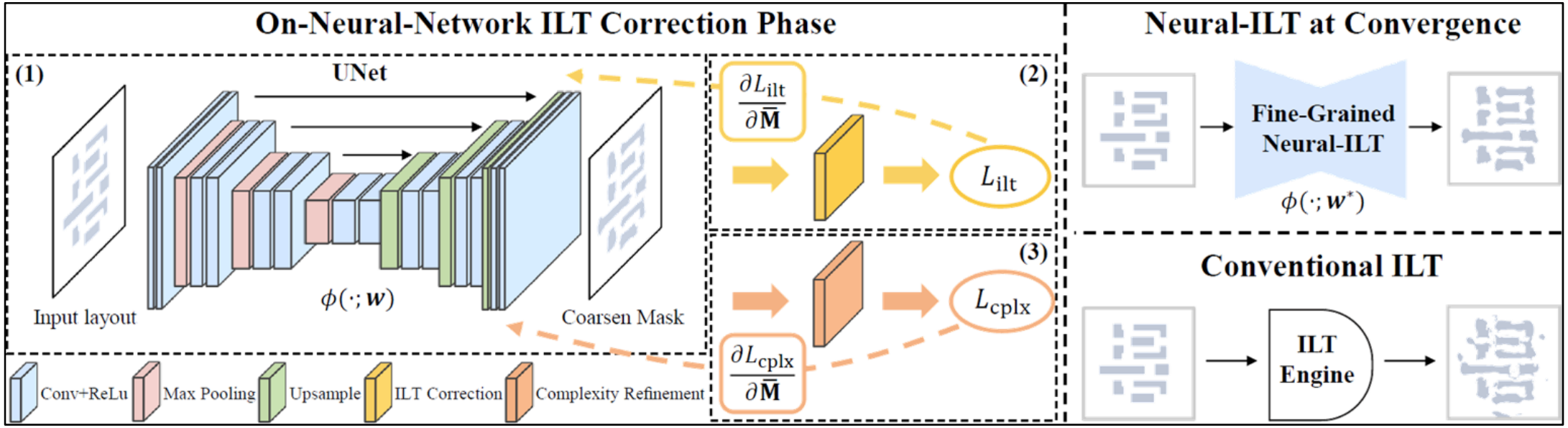
- The on-neural-network ILT correction is essentially an **unsupervised training procedure** of Neural-ILT with following objective

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



All in One Network

- **End-to-end** ILT correction with purely learning-based techniques
- Directly generate the masks after ILT without any additional rigorous refinement on the network output



Retrain Backbone with Domain Knowledge

- Original ILT synthesized training dataset usually consist of numerous complex features
  - We use a Neural-ILT to purify the original training instances

- Use the refined dataset to re-train the UNet with the cycle loss L_{cycle}

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

  - Domain knowledge of the ***partially coherent imaging model*** is introduced into the network training
  - ILT is ill-posed, term with domain knowledge serves as a **regularization** term
  - Guide the re-trained network $\phi(\cdot\,; \mathrm{w})$ gradually converged along a domain-specified direction
  - Obtain better initial solution and hence achieve faster convergence

Outline

- Introduction and Background
- Neural-ILT Algorithm
- Result Visualization and Discussion

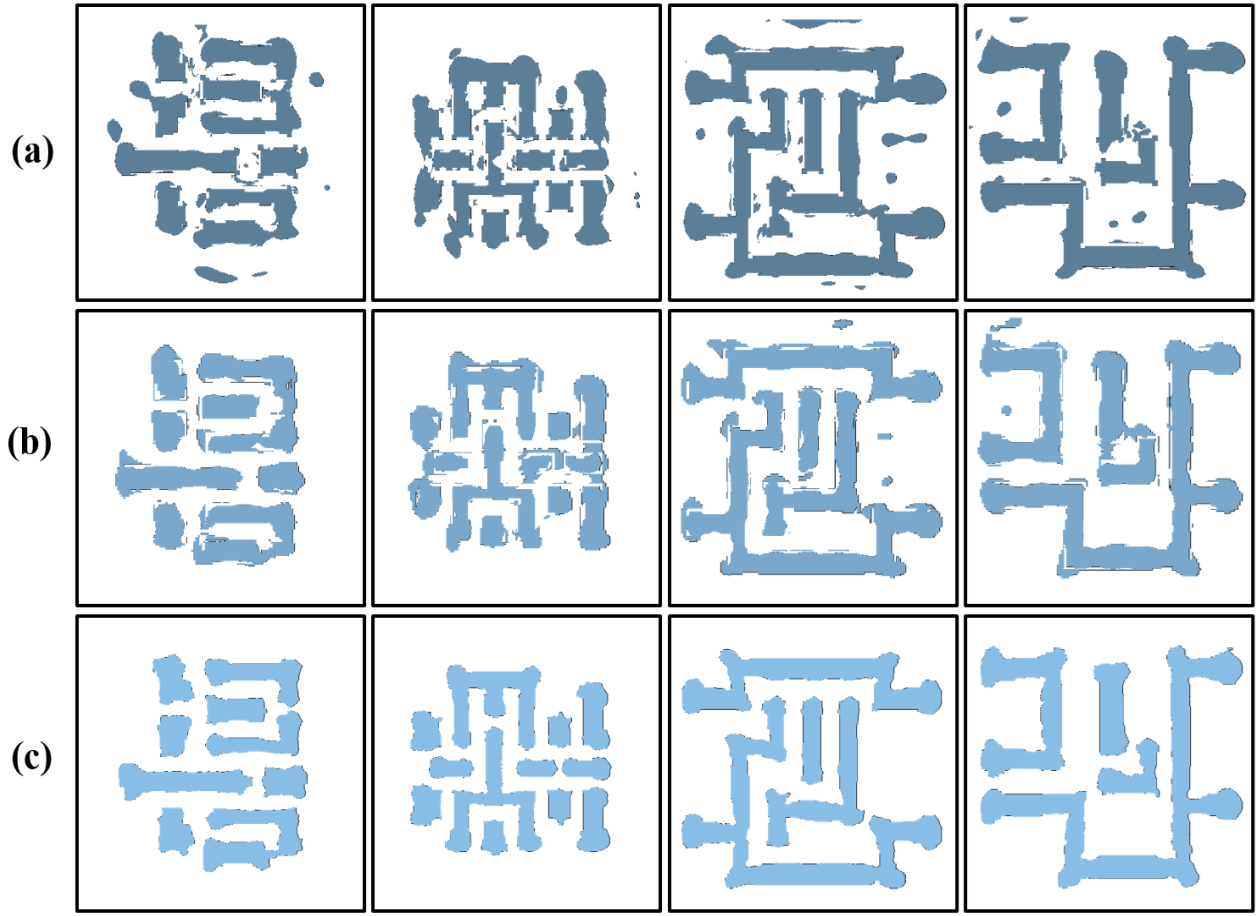
Results

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

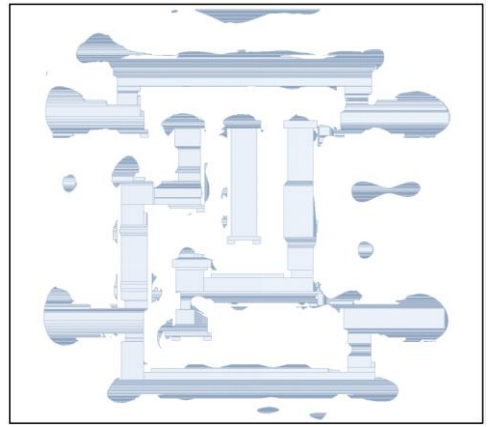
Comparing to SOTA (academia) ILT [4] / PGAN-OPC [5]

- On ICCAD 2013 benchmarks
- **70x, 30x** TAT speedup
- **12.3%**, **3.4%** squared L2 error reduction
- **67%**, **21%** mask fracturing shot count reduction

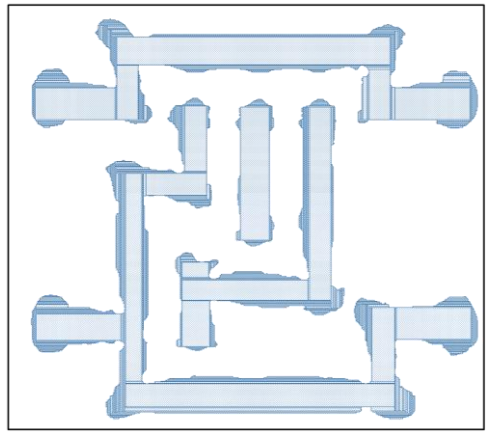
Results



(a) ILT, (b) PGAN-OPC, (c) Neural-ILT



(1) ILT output mask, use 2045 shots to accurately replicate the mask



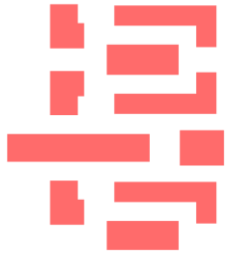
(2) Neural-ILT output mask, use 653 shots to accurately replicate the mask

Animation: Neural-ILT vs. Conventional ILT

Learning rate (stepsize)

- Neural-ILT is decreasing from **1e-3**
- Convectional ILT is decreasing from **1.0**

Mask



Target pattern

Neural-ILT correction process

Runtime = 13.57 secs

Wafer

Target pattern

Mask

Iteration = 01

ILT correction process

Runtime = 1280 secs

Wafer

Iteration = 01

Better Initial Solution and Convergence

Initial Solution

Solution after 20 iterations

Mask

Mask

Mask

Mask

Neural-ILT correction process

Runtime = 13.57 secs

ILT correction process

Runtime = 1280 secs

Neural-ILT correction process

Runtime = 13.57 secs

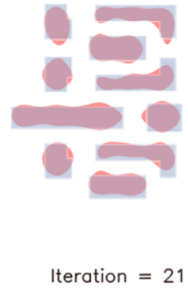
ILT correction process

Runtime = 1280 secs

Wafer

Wafer

Wafer

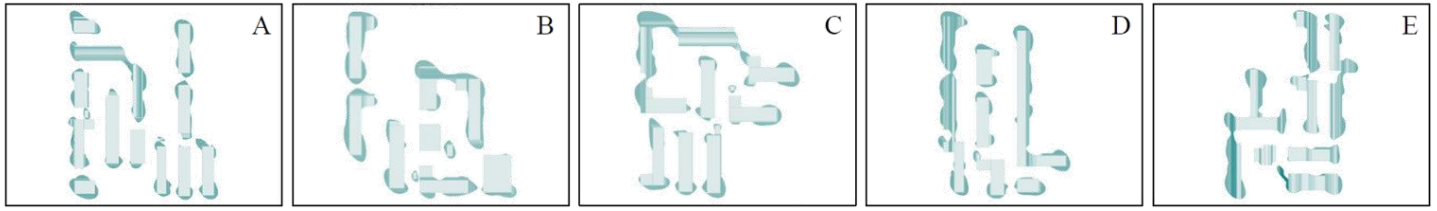


Wafer

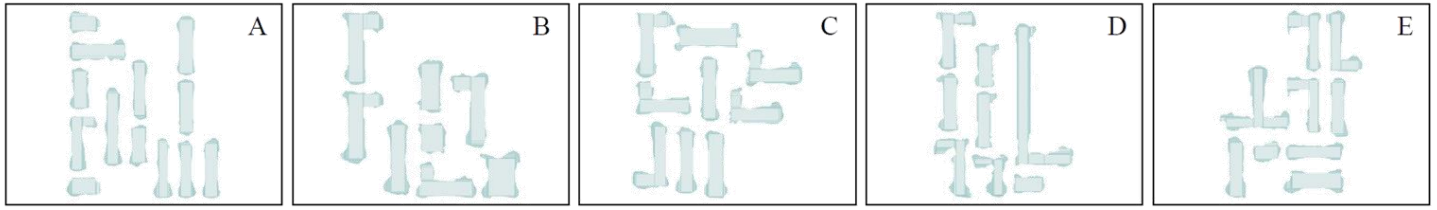
- The initial solution of Neural-ILT has much better printability (smaller image errors)
- May lead to faster and better convergence

Why Neural Network – Empirical Observation

- GPU-ILT v.s. Neural-ILT, Neural-ILT enjoys
  - Higher searching efficiency: less ILT iterations (i.e., 100 vs. 40)
  - Smooth and fine-grained search: much smaller learning rate (i.e., 1.0 vs. 0.001)
  - Larger searching space: better overall quality (i.e, 9% better printability, 51% less shots counts)
- Reserved inverse lithography function
  - Original ILT loses every internal steps except the final $\mathrm{M}_{\mathrm{opt}}$
  - Converged Neural-ILT is indeed an (approximated) inverse lithography function $f^{-1}(\cdot\,;\cdot)$ for the given target layout



(a) Direct mask fracturing [20] results of GPU-ILT synthesized masks on benchmarks A-E.



(b) Direct mask fracturing [20] results of Neural-ILT synthesized masks on benchmarks A-E.

End

Reference

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

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

[3] B. Jiang, X. Zhang, R. Chen, G. Chen, P. Tu, W. Li, E. F. Young, and B. Yu, “Fit: Fill insertion considering timing,” in *Proc. DAC*, 2019, p.221

[4] 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, pp. 52:1–52:6

[5] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, “GAN-OPC: Mask optimization with lithography-guided generative adversarial nets,” in Proc. DAC, 2018, pp. 131:1–131:6