

## Differentiable Computational Lithography Framework

Guojin Chen<sup>1,2</sup>, Hao Geng<sup>3</sup>, Bei Yu<sup>1</sup>, David Z. Pan<sup>2</sup>

<sup>1</sup>CUHK, <sup>2</sup>UT Austin, <sup>3</sup>ShanghaiTech  
[gjchen@utexas.edu](mailto:gjchen@utexas.edu)

March 6, 2024



**TEXAS**  
The University of Texas at Austin



## 1 Background

- 1.1 Differentiable Programming
- 1.2 Lithography Simulation

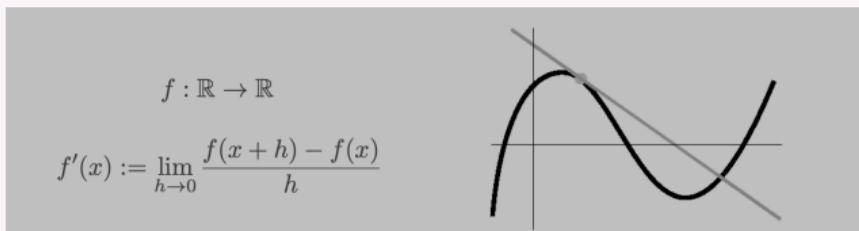
## 2 Differentiable Lithography

- 2.1 Lithography Modeling
- 2.2 Implementation of Differentiable Lithography
- 2.3 Composable Differentiable Lithography

# Differentiable Programming

# Remember derivatives and gradients?

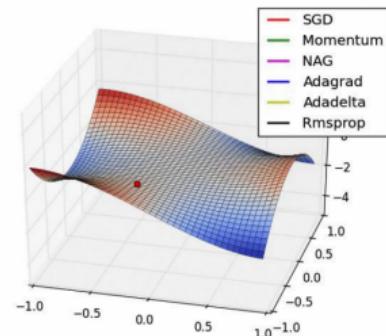
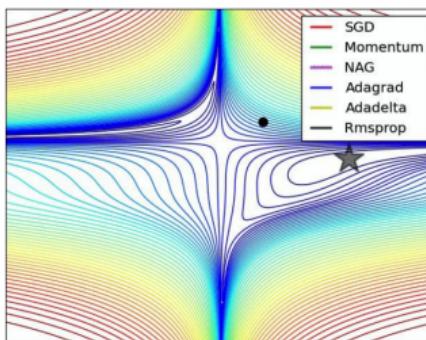
## Derivative



## Gradient

$$Q(\mathbf{w}) = \sum_{i=1}^N Q_i(\mathbf{w})$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{i=1}^d \nabla_{\mathbf{w}} Q_i(\mathbf{w})$$



(Ruder, 2017) <http://ruder.io/optimizing-gradient-descent/>

**Automatic Differentiation** : careful application of *chain rule to programs*.

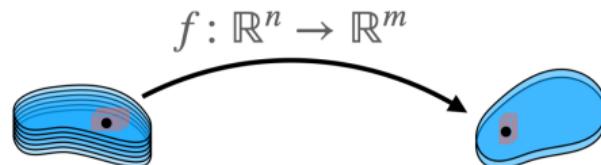
```
import jax
import jax.numpy as jnp

def func(x):
    y = x
    for i in range(4):
        y += x[0]**2 + jnp.sin(x[1]) + jnp.exp(-x[2])
    y = y.sum()
    return y
```

exact gradients!



```
gfunc = jax.value_and_grad(func)
gfunc(jnp.array([2.,3.,-2]))
(DeviceArray(141.36212, dtype=float32),
 DeviceArray([ 49.          , -10.8799095, -87.66867  ], dtype=float32))
```



$$y = f(x) \quad dy = J_f dx$$

 TensorFlow

 JAX

 PYTORCH

... but also C++, Fortran, ...

$$J_f = \frac{\partial(y_1, \dots, y_m)}{\partial(x_1, \dots, x_n)}$$

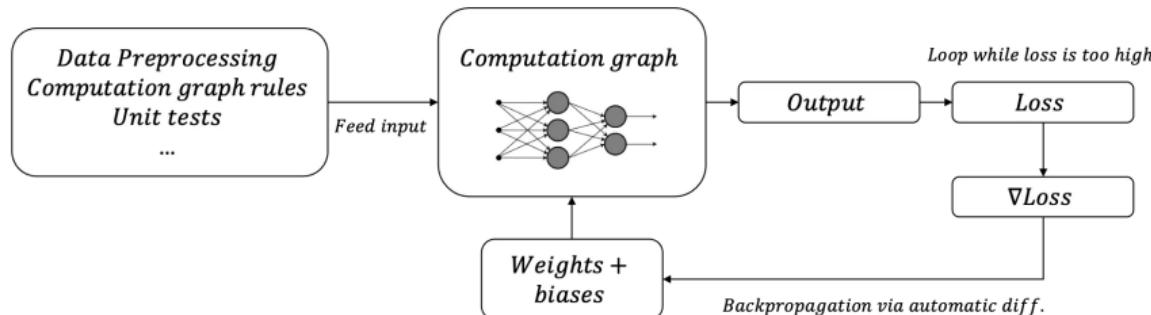
Automatic Differentiation = methods for automatically computing gradients of functions specified by a computer program.

## Execute differentiable code via **automatic differentiation**.

**Differentiable programming:** Writing software composed of **differentiable and parameterized building blocks** that are executed via **automatic differentiation** and **optimized** in order to perform a specified task.

- ① A **parameterized function** (method / model / building blocks) to be optimized;
- ② Automatic differentiability of the function to be **optimized**.
- ③ A **loss** to measure performance;

differentiable programming = programming languages + *automatic differentiation*.





Yann LeCun

January 5, 2018 ·

OK, Deep Learning has outlived its usefulness as a buzz-phrase.  
Deep Learning est mort. Vive Differentiable Programming!



Andrej Karpathy

@karpathy · [Follow](#)



Gradient descent can write code better than you. I'm sorry.

3:56 PM · Aug 4, 2017



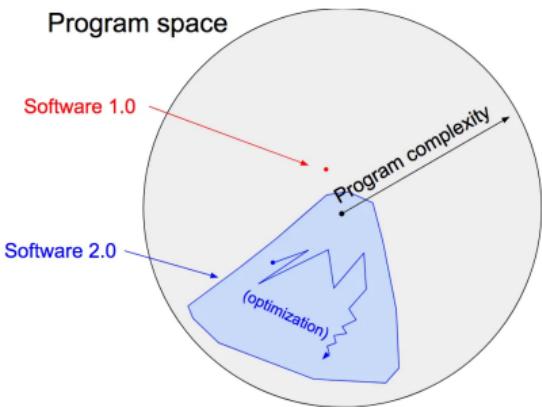
2.7K



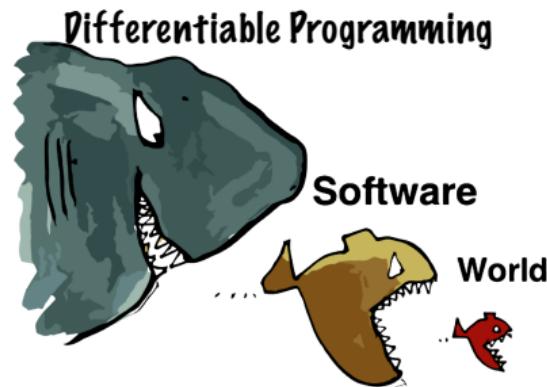
Reply



Share



Software 2.0 from Andrej Karpathy<sup>1</sup>

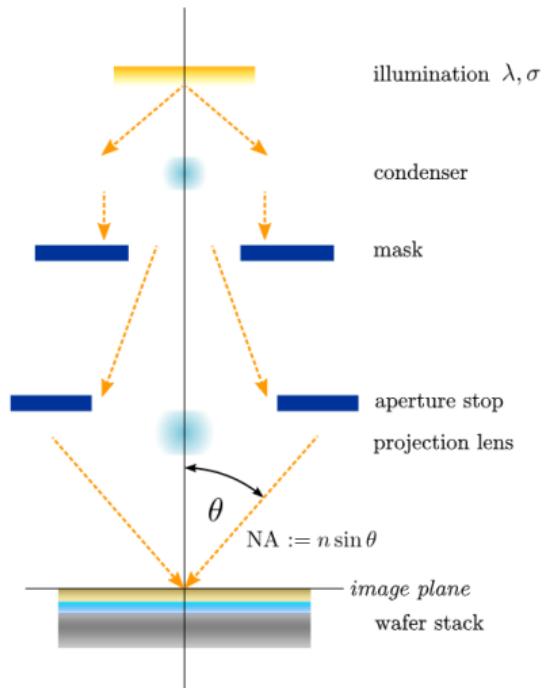


AI is eating software from Jensen Huang<sup>2</sup>

<sup>1</sup>Software 2.0: <https://karpathy.medium.com/software-2-0-a64152b37c35>

<sup>2</sup>AI: <https://www.technologyreview.com/ai-is-going-to-eat-software/>

# Background of Lithography



3

<sup>3</sup>Tim Fühner. "Artificial Evolution for the Optimization of Lithographic Process Conditions". In: 2014.

# The scalar imaging equation

The scalar imaging equation under partially coherent illumination

$$\begin{aligned} I(x_1, y_1) &= J_I((x_1, y_1), (x_1, y_1)) \\ &= \iiint \int_{-\infty}^{\infty} J_C(x_0 - x'_0, y_0 - y'_0) O(x_0, y_0) O^*(x'_0, y'_0) \\ &\quad H(x_1 - x_0, y_1 - y_0) H^*(x_1 - x'_0, y_1 - y'_0) dx_0 dy_0 dx'_0 dy'_0, \end{aligned} \quad (1)$$

where  $O$  is the object function, the field of the photomask in the lithography case,  $H$  is the projector transfer function, and  $J_C$  is the mutual intensity, a weight factor, of two points under extended source conditions.

## Conclusion

The intensity at a point in the image plane is given by the propagation of the mutual intensity of all contributing points, that is, of all points that lay in the support of the projection system and the illuminator.

- Abbe's approach

- illumination cross-coefficients (**ICC**)

$$ICC(x, y; f, g) = \left| \iint_{-\infty}^{\infty} H(f + f', g + g') \mathcal{F}(M)(f', g') \exp(-j2\pi(f'x + g'y)) df' dg' \right|^2.$$

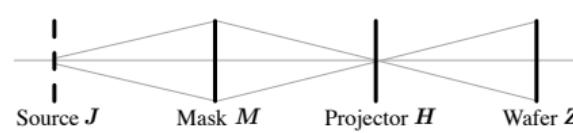
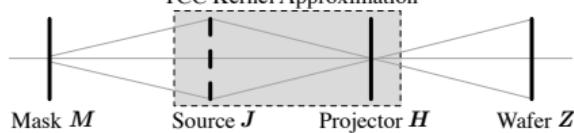
- Abbe's approach

$$I(x, y) = \iint_{-\infty}^{\infty} J(f, g) ICC(x, y; f, g) df dg.$$

- Hopkins' approach

- TCC

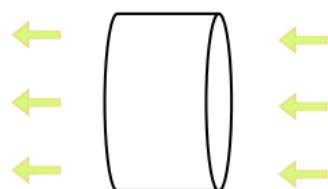
$$I(x, y) = \iiint_{-\infty}^{\infty} \mathcal{T}(f', g'; f'', g'') \mathcal{F}(M)(f', g') \mathcal{F}(M)^*(f'', g'') \exp(-j2\pi((f' - f'')x + (g' - g'')y)) df' dg' df'' dg'',$$

	Abbe	Hopkins
Visualization		
Complexity	$\mathcal{O}(n^6)$ Can be accelerated using parallel computing, or compressive sensing.	$\mathcal{O}(n^4)$ Accelerated by SOCS. Can be further accelerated using GPU parallel computing.
Application	Source optimization. Source Mask Co-optimization.	Fast lithographic simulation. Mask optimization.

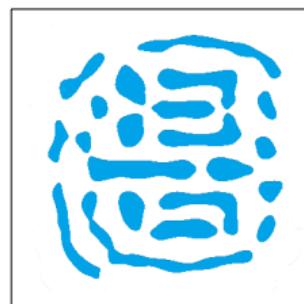
# What's next?



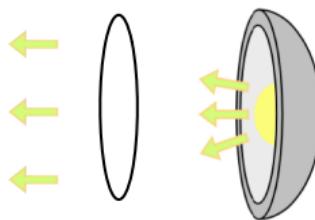
Wafer



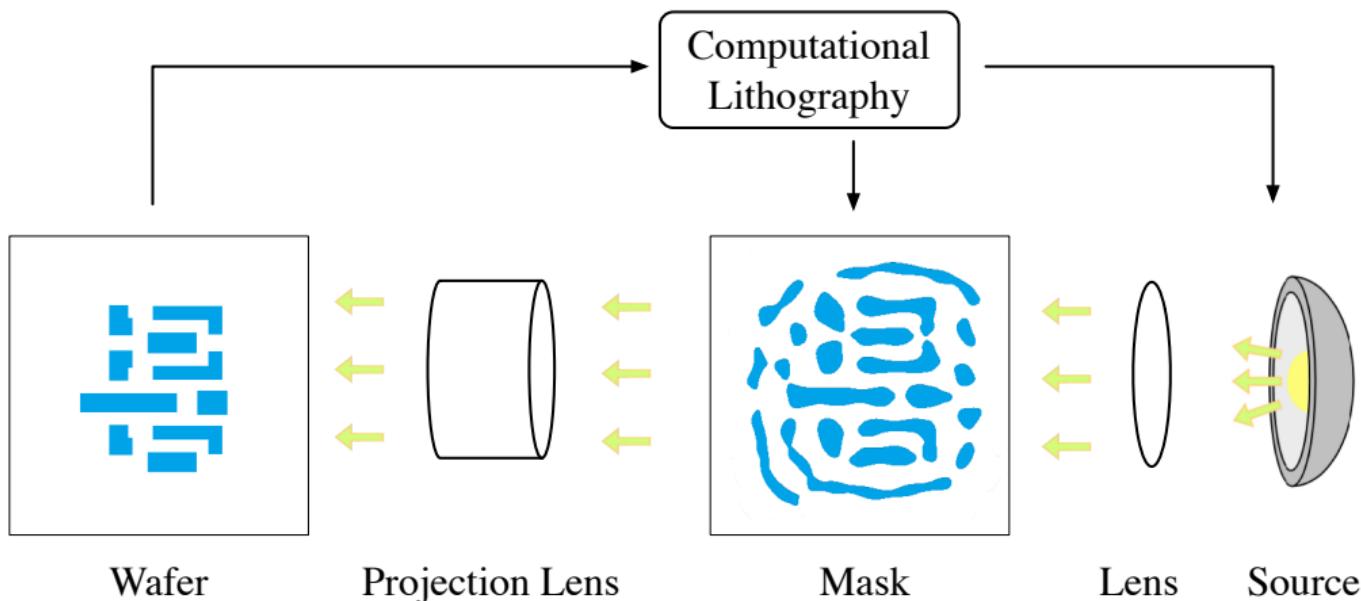
Projection Lens



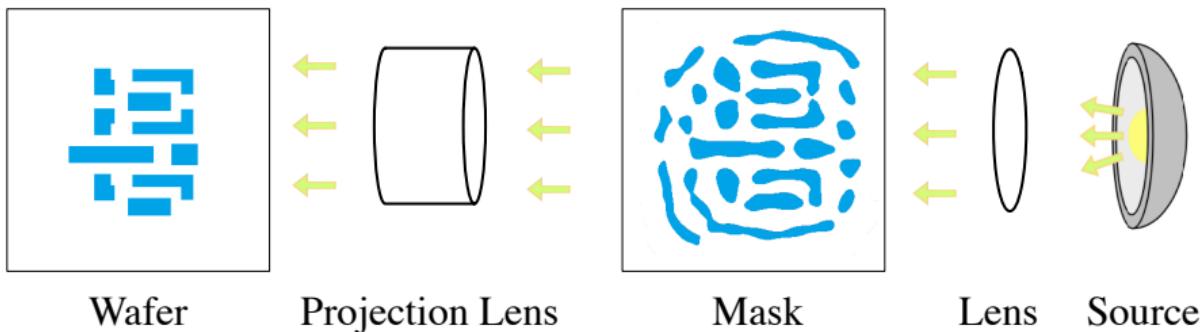
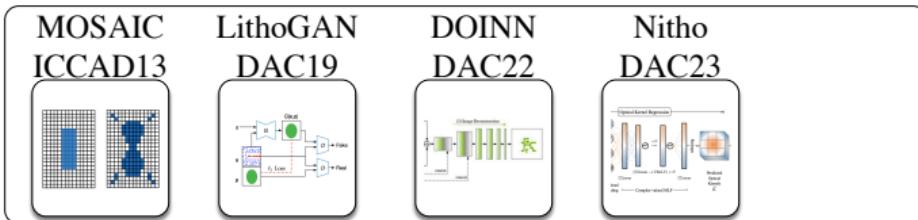
Mask

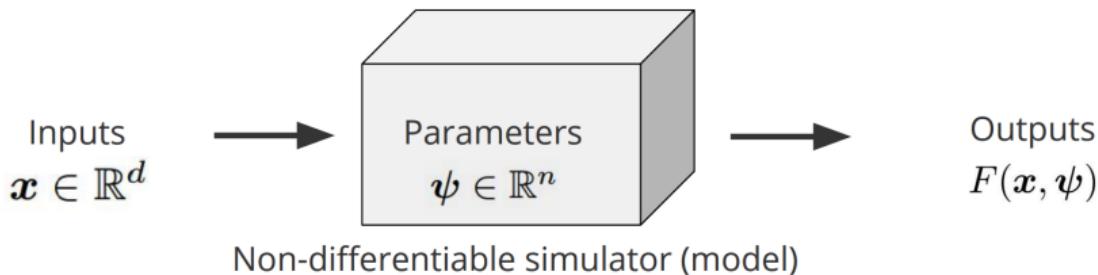


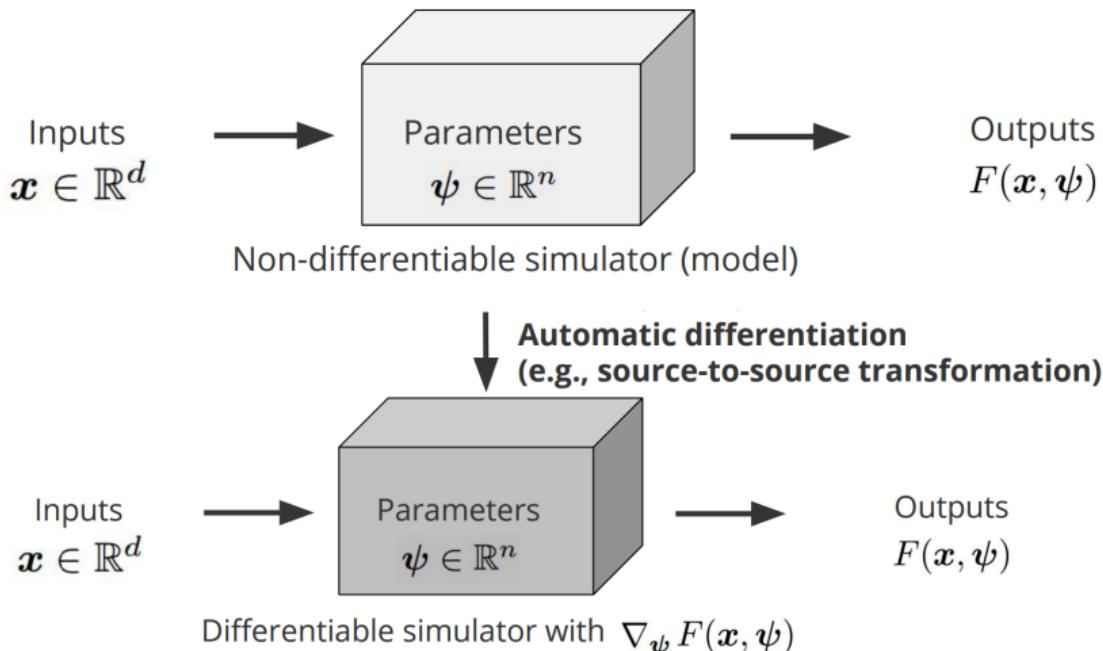
Lens      Source



## Lithography Modeling



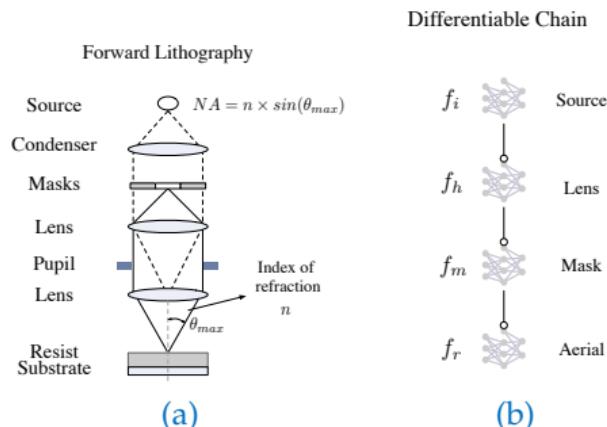




- Use automatic differentiation tools to make the simulator directly differentiable.

## How can we implement differentiable lithography?

- Complex lithography setups can be composed of a pipeline of a series of distinct modules *i.e.*, source, lens, mask, aerial.
- One might need to differentiate through the whole end-to-end pipeline, which can be achieved by compositionality and the chain rule.

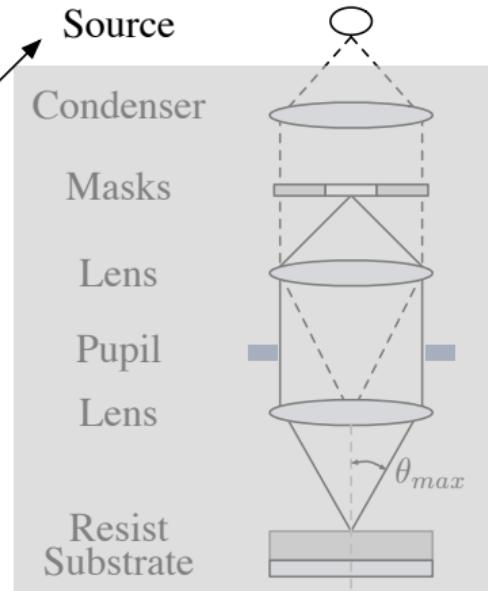


(a) Core components of forward lithography process. (b) The visualization of the differentiable lithography chain.

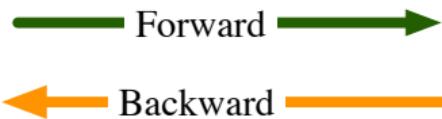
- **Differentiable analysis**
  - Unify analysis pipeline by simultaneously optimizing the free parameters of an analysis with respect to the desired physics objective.
- **Differentiable simulation**
  - Enable efficient simulation-based inference, reducing the number of events needed by orders of magnitude.

# Differentiable Source Module

```
class Source:  
    """Source.data is used for Abbe formulation  
    Source.mdata is used for Hopkins formulation, Mutual  
    Intensity, TCC calculation."""  
  
    def __init__(  
        self,  
        na: float = 1.35,  
        wavelength: float = 193.0,  
        maskpitch: float = 2000.0,  
        maskypitch: float = 2000.0,  
        sigma_out: float = 0.8,  
        sigma_in: float = 0.6,  
        smooth_data: float = 0.03,  
        source_type: str = "annular",  
        shiftAngle: float = math.pi / 4,  
        openAngle: float = math.pi / 16,  
    ):  
        self.na = na  
        self.wavelength = wavelength  
        self.maskpitch = maskpitch  
        self.maskypitch = maskypitch  
  
        self.sigma_out = sigma_out  
        self.sigma_in = sigma_in  
        self.smooth_data = smooth_data  
        self.shiftAngle = shiftAngle  
        self.openAngle = openAngle  
        self.type = source_type  
  
    ***  
    Process calculation  
    ***  
    self.update()
```



1. Init Parameters
  2. Calculate source



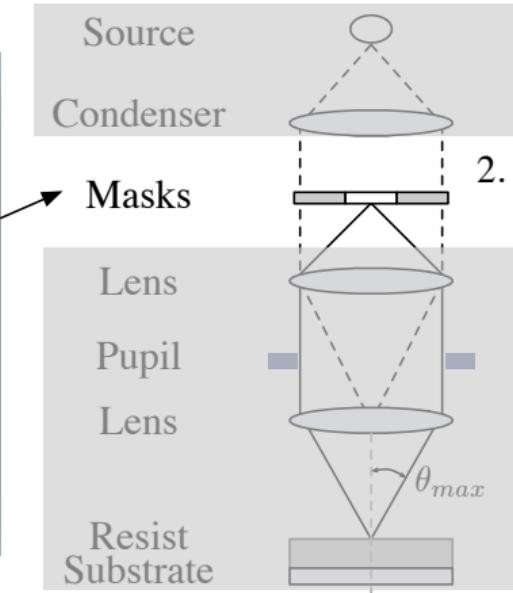
Calculate source gradient  
with respect to source value

<sup>3</sup><https://github.com/TorchOPC/TorchLitho>

# Differentiable Mask Module

```
class Mask:
    def __init__(self,
                 gds_path: str,
                 layername: int = 11,
                 pixels_per_um: int = 100,
                 xmax=1024,
                 ymax=1024,
                 x_grayscale=1,
                 y_grayscale=1,
                 CD=45,
                 ):
        self.x_range = [-xmax, xmax] # nm
        self.y_range = [-ymax, ymax]
        self.x_grayscale = x_grayscale
        self.y_grayscale = y_grayscale
        self.mask_groups = []
        self.CD = CD
        self.gds_path = gds_path
        self.layername = layername
        self.pixels_per_um = pixels_per_um

    """
    Process calculation
    """
    self.openGDS()
    self.maskfft()
```



1. Init mask layout / params
2. Calculate mask spectrum / value

Forward

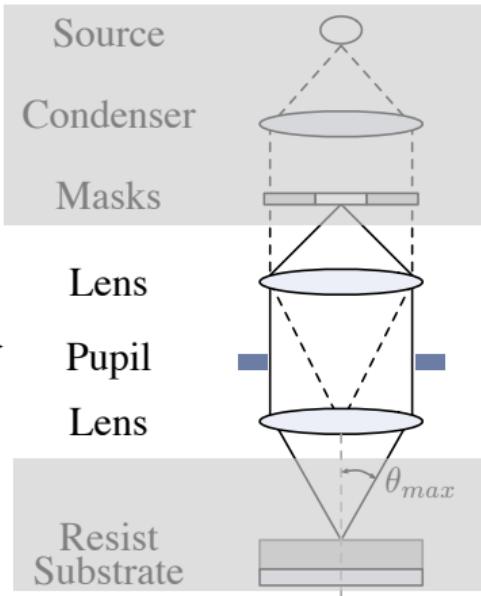
Backward

Calculate gradient  
with respect to mask params

<sup>3</sup><https://github.com/TorchOPC/TorchLitho>

# Differentiable Lens Module

```
class LensList(Lens):
    """List of lens."""
    def __init__(self, na: float = 1.35, nliquid: float = 1.46, wavelength: float = 193.0, defocus: float = 0.0, maskxpitch: float = 1000, maskypitch: float = 1000,):
        super().__init__(na, nliquid, wavelength, defocus, maskxpitch, maskypitch)
        self.fFocusList = [0.0]
        self.fFocusCef = [1.0]
        self.fDataList = []
        self.sDataList = []
```



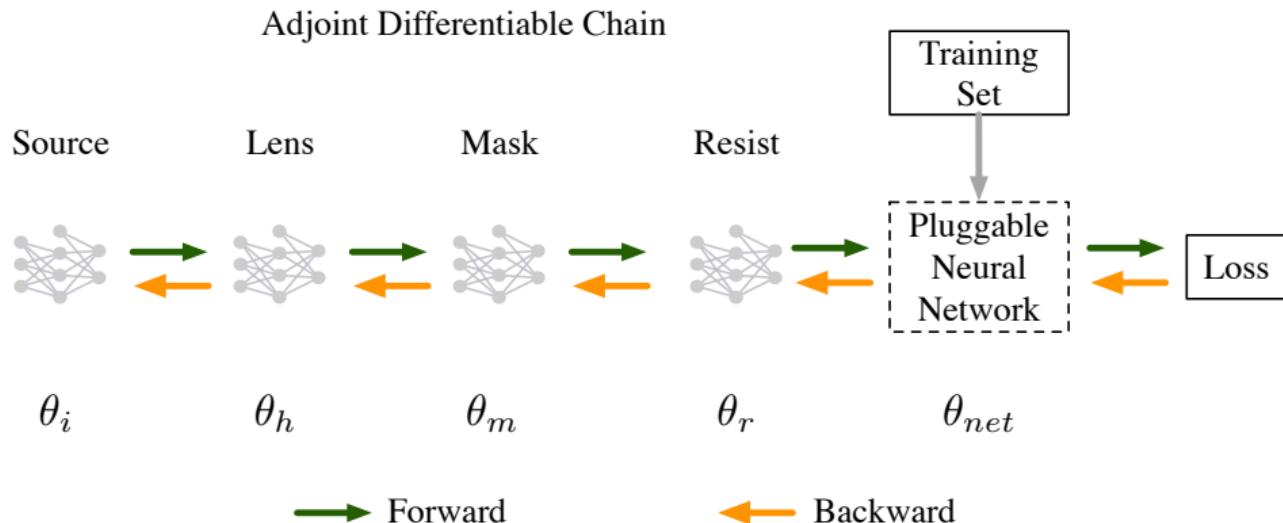
1. Init projection parameters
2. Calculate PSF / TCC

Forward

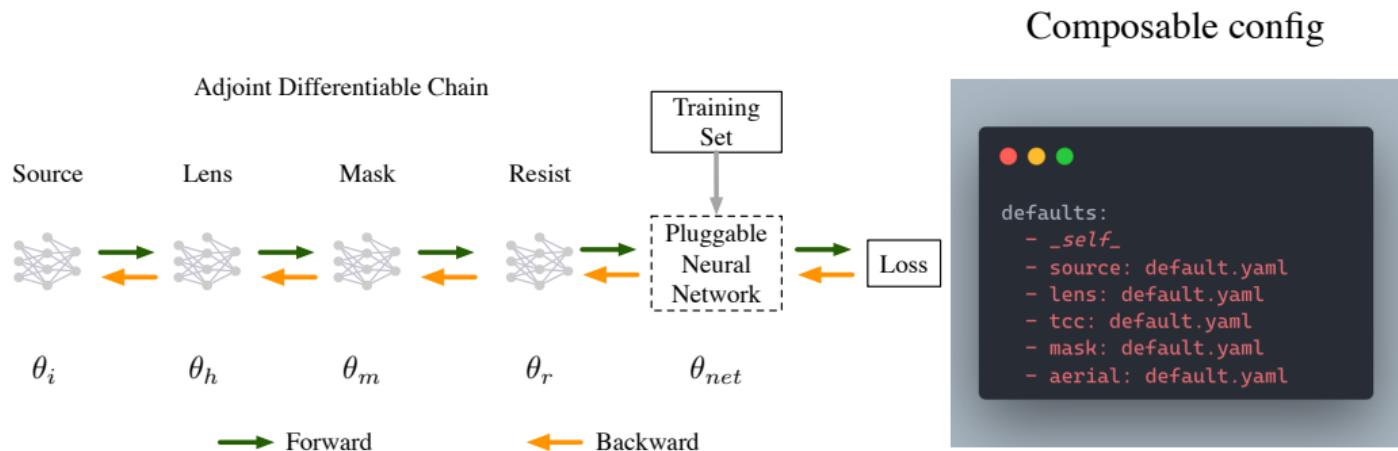
Backward

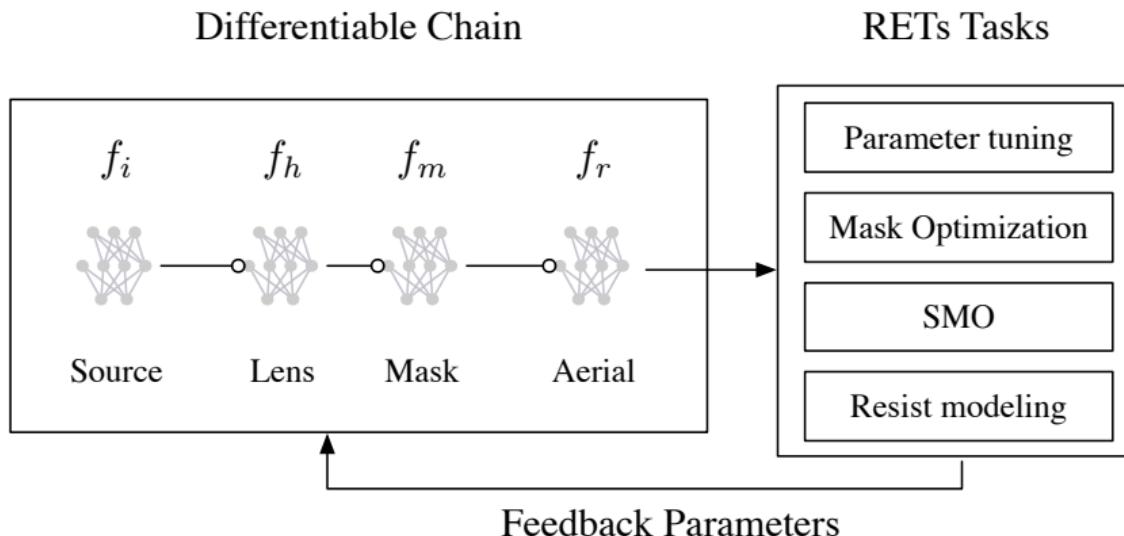
Calculate gradient  
with respect to lens params

<sup>3</sup><https://github.com/TorchOPC/TorchLitho>



# Composable Differentiable Lithography





# Multi-level optimization framework.

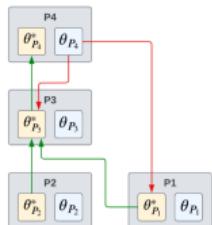
$$P_n : \quad \theta_n^* = \underset{\theta_n}{\operatorname{argmin}} \mathcal{C}_n(\theta_n, \mathcal{U}_n, \mathcal{L}_n; \mathcal{D}_n) \quad \triangleright \text{Level } n \text{ problem}$$

⋮

$$P_k : \quad \text{s.t. } \theta_k^* = \underset{\theta_k}{\operatorname{argmin}} \mathcal{C}_k(\theta_k, \mathcal{U}_k, \mathcal{L}_k; \mathcal{D}_k) \quad \triangleright \text{Level } k \in \{2, \dots, n-1\} \text{ problem}$$

⋮

$$P_1 : \quad \text{s.t. } \theta_1^* = \underset{\theta_1}{\operatorname{argmin}} \mathcal{C}_1(\theta_1, \mathcal{U}_1, \mathcal{L}_1; \mathcal{D}_1) \quad \triangleright \text{Level 1 problem}$$



Source Optimization : optimize source parameters, fix others.

Mask Optimization : optimize mask parameters, fix others.

Source Mask Optimization : bi-level optimization for source and mask

<sup>3</sup><https://github.com/TorchOPC/TorchLitho>

Dataset	DAMO <sup>21</sup>		TEMPO <sup>34</sup>		DOINN <sup>32</sup>		Ours	
	mPA	mIOU	mPA	mIOU	mPA	mIOU	mPA	mIOU
Benchmark1 <sup>33</sup>	95.2	91.1	94.6	88.7	99.19	98.32	99.45	99.21
Benchmark2 <sup>35</sup>	98.97	97.31	98.24	96.55	98.79	97.1	99.15	99.02
Benchmark3 <sup>35</sup>	99.11	93.56	99.06	93.28	99.21	98.41	99.59	99.34
Benchmark4 <sup>33, 35</sup>	99.01	97.1	98.63	95.84	98.71	96.68	99.61	99.36
Average	98.07	94.77	97.63	93.59	98.98	97.63	<b>99.45</b>	<b>99.23</b>
Ratio	0.99	0.96	0.98	0.94	0.99	0.98	<b>1</b>	<b>1</b>

The comparison of the proposed method and the SOTA method.

- [1] Tim Fühner. "Artificial Evolution for the Optimization of Lithographic Process Conditions". In: 2014.

**THANK YOU!**