SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

Differentiable Computational Lithography Framework

Guojin Chen[1,2], Hao Geng[3], Bei Yu[1], David Z. Pan[2]

[1]CUHK, [2]UT Austin, [3]ShanghaiTech

gjchen@utexas.edu

March 7, 2024

TEXAS
The University of Texas at Austin

Outline

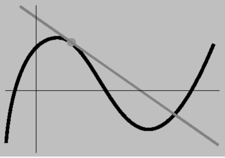
Differentiable Programming

Remember derivatives and gradients?

Derivative

$$f : \mathbb{R} \to \mathbb{R}$$

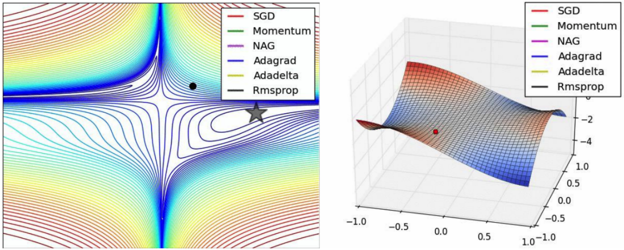
$$f'(x) := \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



Gradient

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

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



(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! ↓

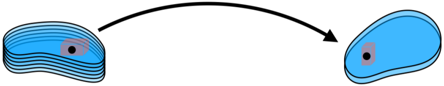
```
gfunc = jax.value_and_grad(func)
gfunc(jnp.array([2.,3.,-2]))

(DeviceArray(141.36212, dtype=float32),
 DeviceArray([ 49.        , -10.8799095, -87.66867   ], dtype=float32))
```

TensorFlow JAX PYTORCH

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

$f : \mathbb{R}^n \to \mathbb{R}^m$



$$y = f(x) \qquad dy = J_f\, dx$$

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

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

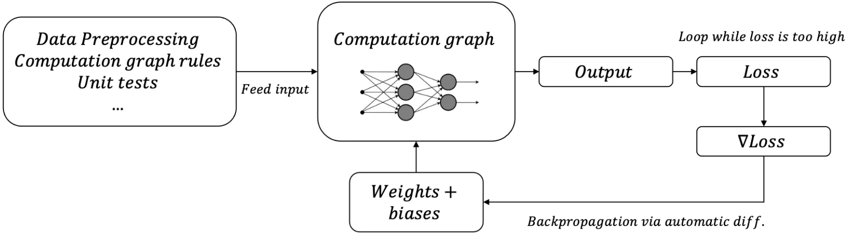
SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

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`.



Differentiable programming: master quotes

SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

Yann LeCun
January 5, 2018 ·

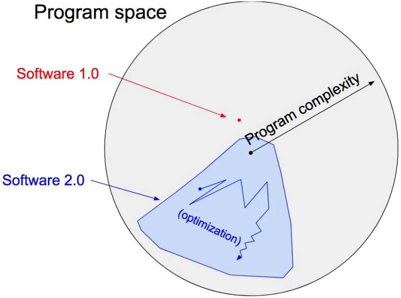
OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Andrej Karpathy
@karpathy · Follow

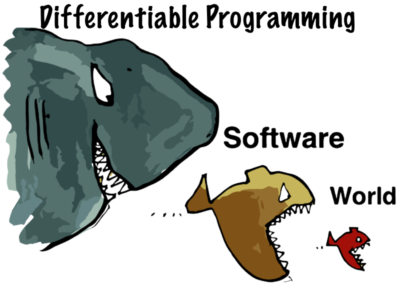
Gradient descent can write code better than you. I'm sorry.

3:56 PM · Aug 4, 2017

2.7K Reply Share



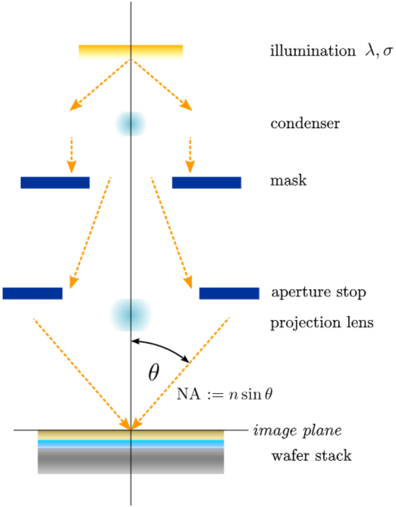
Software 2.0 from Andrej Karpathy[1]



AI is eating software from Jensen Huang[2]

[2]Software 2.0: https://karpathy.medium.com/software-2-0-a64152b37c35
[2]AI: https://www.technologyreview.com/ai-is-going-to-eat-software/

Background of Lithography



3

[3]Tim Führner. "Artificial Evolution for the Optimization of Lithographic Process Conditions". In: 2014.

SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

The scalar imaging equation under partially coherent illumination

$$
\begin{aligned}
I(x_1, y_1) &= J_I((x_1, y_1), (x_1, y_1)) \\
&= \iiint \int_{-\infty}^{\infty} J_C(x_0 - x_0', y_0 - y_0') \, O(x_0, y_0) \, O^*(x_0', y_0') \\
& H(x_1 - x_0, y_1 - y_0) \, H^*(x_1 - x_0', y_1 - y_0') \, \mathrm{d}x_0 \, \mathrm{d}y_0 \, \mathrm{d}x_0' \mathrm{d}y_0',
\end{aligned} \tag{1}
$$

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.

SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

- **Abbe's approach**
  - *illumination cross-coefficients* (***ICC***)

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

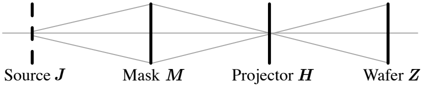
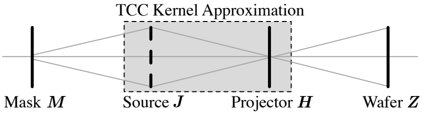
  - Abbe's approach

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

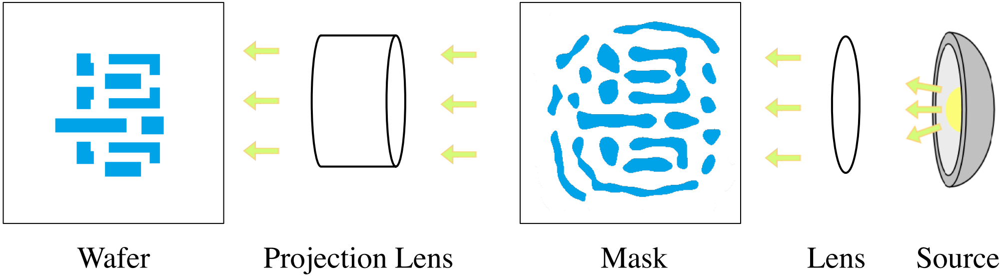
- **Hopkins' approach**
  - TCC

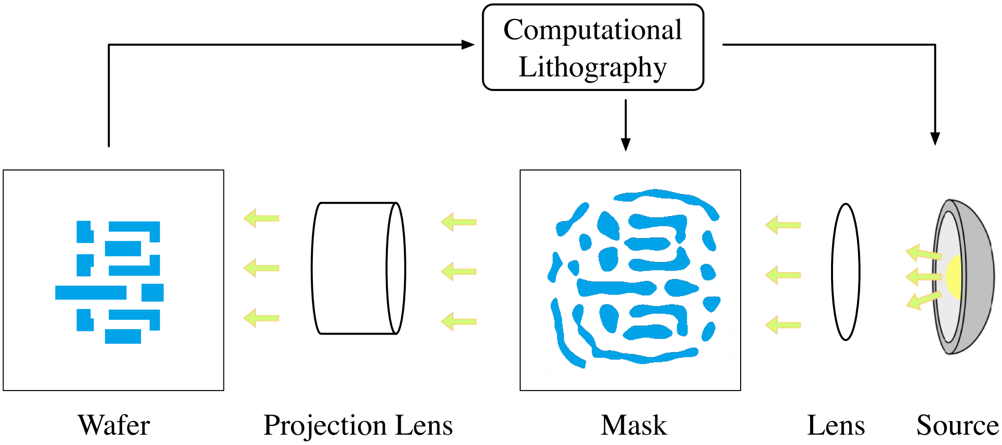
$$I(x,y) = \iiiint_{-\infty}^{\infty} \mathcal{T}(f',g';f'',g'')\mathcal{F}(M)(f',g')\mathcal{F}(M)^*(f'',g'') \exp(-j2\pi((f'-f'')x+(g'-g'')y))\mathrm{d}f'\mathrm{d}g'\mathrm{d}f''\mathrm{d}g'',$$

Abbe's VS Hopkins'

	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.

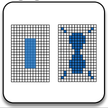
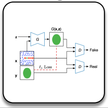
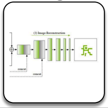
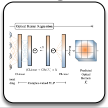
What’s next?

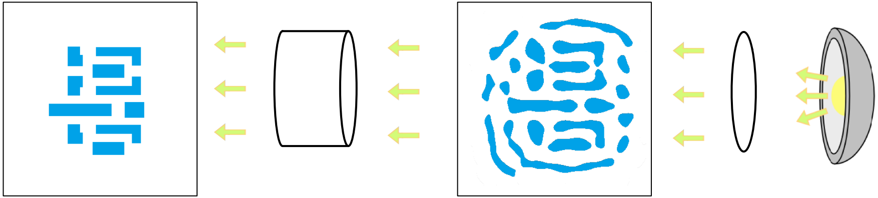

Wafer
Projection Lens
Mask
Lens
Source

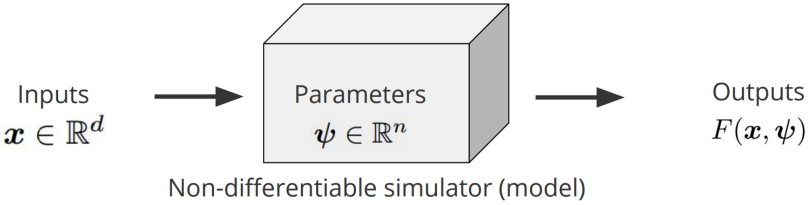

Computational Lithography
Wafer
Projection Lens
Mask
Lens
Source

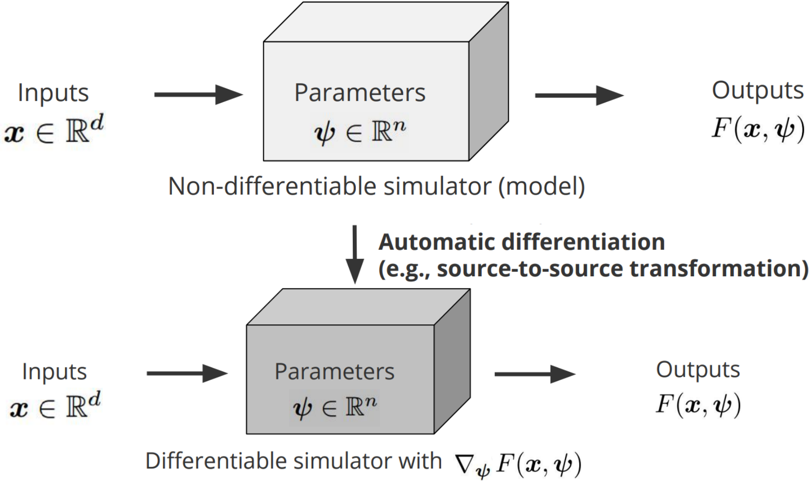
Lithography Modeling

Lithography Modeling

MOSAIC ICCAD13	LithoGAN DAC19	DOINN DAC22	Nitho DAC23
			



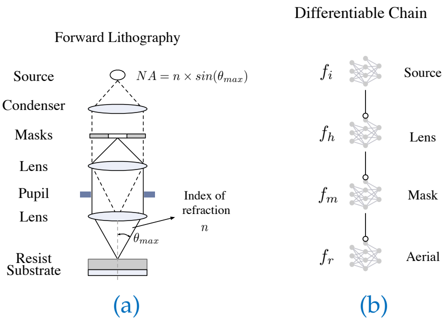

Inputs
$\boldsymbol{x} \in \mathbb{R}^d$
Parameters
$\boldsymbol{\psi} \in \mathbb{R}^n$
Outputs
$F(\boldsymbol{x}, \boldsymbol{\psi})$
Non-differentiable simulator (model)



- 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.

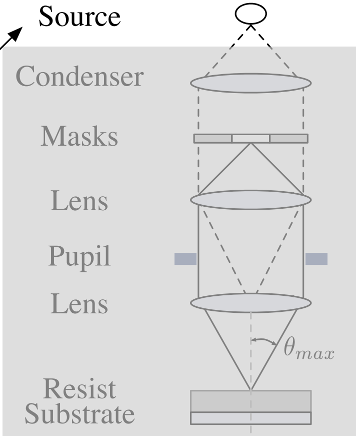
Differentiable Source Module

```python
class Source:
    """Source.data is used for Abbe fomulation
Source.mdata is used for Hopkins fomulation, Mutual
    Intensity, TCC calculation."""

    def __init__(
        self,
        na: float = 1.35,
        wavelength: float = 193.0,
        maskxpitch: float = 2000.0,
        maskypitch: float = 2000.0,
        sigma_out: float = 0.8,
        sigma_in: float = 0.6,
        smooth_deta: float = 0.03,
        source_type: str = "annular",
        shiftAngle: float = math.pi / 4,
        openAngle: float = math.pi / 16,
    ):
        self.na = na
        self.wavelength = wavelength
        self.maskxpitch = maskxpitch
        self.maskypitch = maskypitch

        self.sigma_out = sigma_out
        self.sigma_in = sigma_in
        self.smooth_deta = smooth_deta
        self.shiftAngle = shiftAngle
        self.openAngle = openAngle
        self.type = source_type

        """
        Process calculation
        """
        self.update()
```



1. Init Parameters
2. Calculate source

Forward →

← Backward

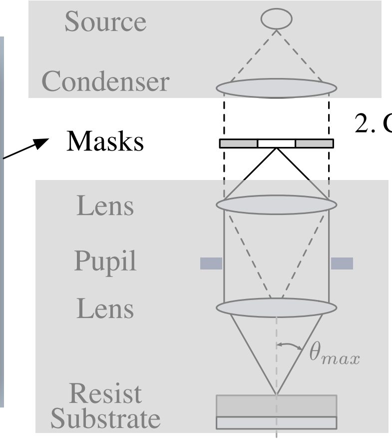
Calculate source gradient with respect to source value

[3] https://github.com/TorchOPC/TorchLitho

Differentiable Mask Module

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

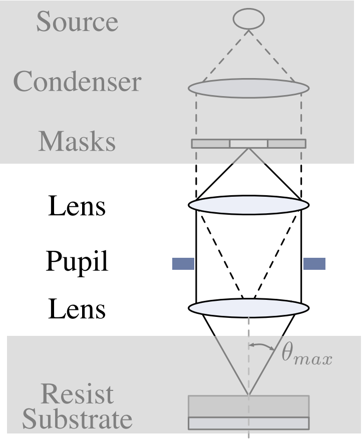
Calculate gradient
with respect to mask params

[3] https://github.com/TorchOPC/TorchLitho

Differentiable Lens Module

```python
class LensList(Lens):
    """List of lens."""

    def __init__(
        self,
        na: float = 1.35,
        nLiquid: float = 1.44,
        wavelength: float = 193.0,
        defocus: float = 0.0,
        maskxpitch: float = 1000,
        maskypitch: float = 1000,
    ):
        super().__init__(na, nLiquid, wavelength,
defocus, maskxpitch, maskypitch)
        self.focusList = [0.0]
        self.focusCoef = [1.0]
        self.fDataList = []
        self.sDataList = []
```



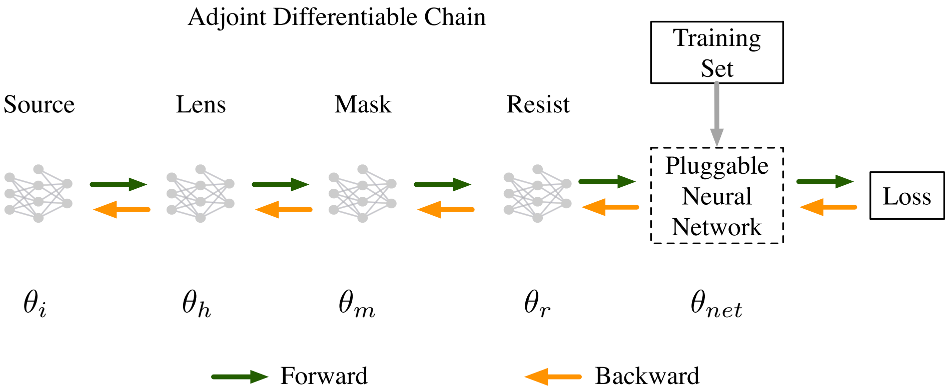
1. Init projection parameters
2. Calculate PSF / TCC

Forward

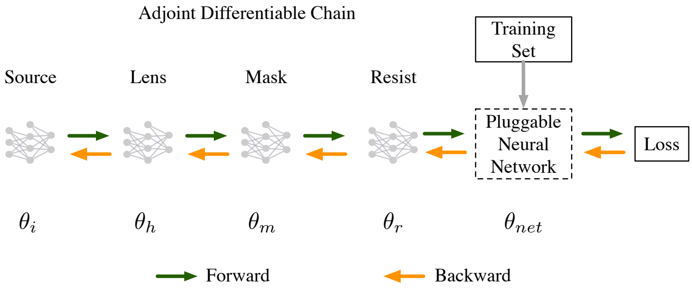
Backward

Calculate gradient
with respect to lens params

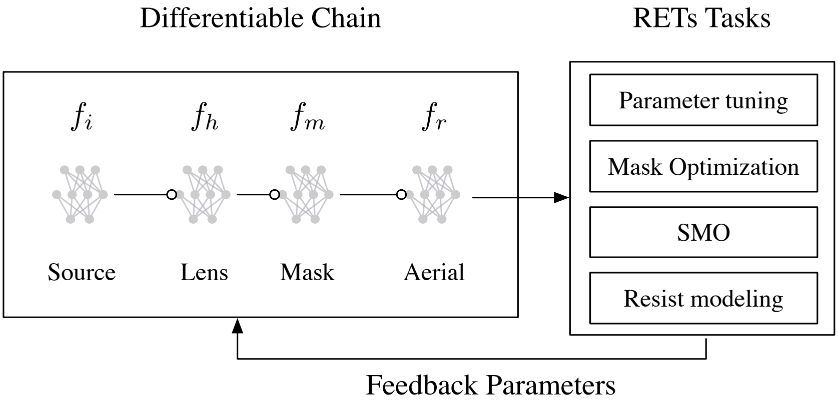
[3] https://github.com/TorchOPC/TorchLitho



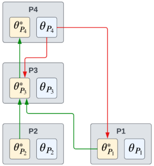
Adjoint Differentiable Chain
Training Set
Source
Lens
Mask
Resist
Pluggable Neural Network
Loss
θ_i
θ_h
θ_m
θ_r
θ_{net}
Forward
Backward



Composable config


Differentiable Chain
RETs Tasks
f_i
f_h
f_m
f_r
Source
Lens
Mask
Aerial
Parameter tuning
Mask Optimization
SMO
Resist modeling
Feedback Parameters

$$
\begin{aligned}
P_n: &\quad \theta_n^* = \operatorname*{argmin}_{\theta_n} \mathcal{C}_n(\theta_n, \mathcal{U}_n, \mathcal{L}_n; \mathcal{D}_n) && \triangleright \text{ Level } n \text{ problem} \\
&\qquad \ddots \\
P_k: &\qquad \text{s.t. } \theta_k^* = \operatorname*{argmin}_{\theta_k} \mathcal{C}_k(\theta_k, \mathcal{U}_k, \mathcal{L}_k; \mathcal{D}_k) && \triangleright \text{ Level } k \in \{2, \ldots, n-1\} \text{ problem} \\
&\qquad\qquad \ddots \\
P_1: &\qquad\qquad \text{s.t. } \theta_1^* = \operatorname*{argmin}_{\theta_1} \mathcal{C}_1(\theta_1, \mathcal{U}_1, \mathcal{L}_1; \mathcal{D}_1) && \triangleright \text{ Level 1 problem}
\end{aligned}
$$



Source Optimization : optimize source parameters, fix others.

Mask Optimization : optimize mask parameters, fix others.

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

[3] https://github.com/TorchOPC/TorchLitho

SPIE. ADVANCED LITHOGRAPHY+ PATTERNING

Dataset	DAMO[21]		TEMPO[34]		DOINN[32]		Ours	
	mPA	mIOU	mPA	mIOU	mPA	mIOU	mPA	mIOU
Benchmark1[33]	95.2	91.1	94.6	88.7	99.19	98.32	99.45	99.21
Benchmark2[35]	98.97	97.31	98.24	96.55	98.79	97.1	99.15	99.02
Benchmark3[35]	99.11	93.56	99.06	93.28	99.21	98.41	99.59	99.34
Benchmark4[33,35]	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	**99.45**	**99.23**
Ratio	0.99	0.96	0.98	0.94	0.99	0.98	**1**	**1**

The comparison of the proposed method and the SOTA method.

References I

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

THANK YOU!