

Methods for Analog Layout Design Automation

Sungyu Jeong

Electrical Engineering Dept.

Pohang University of Science and Technology (POSTECH), South Korea

Advisor: Prof. Byungsub Kim (BEVIL Lab)

Contact:

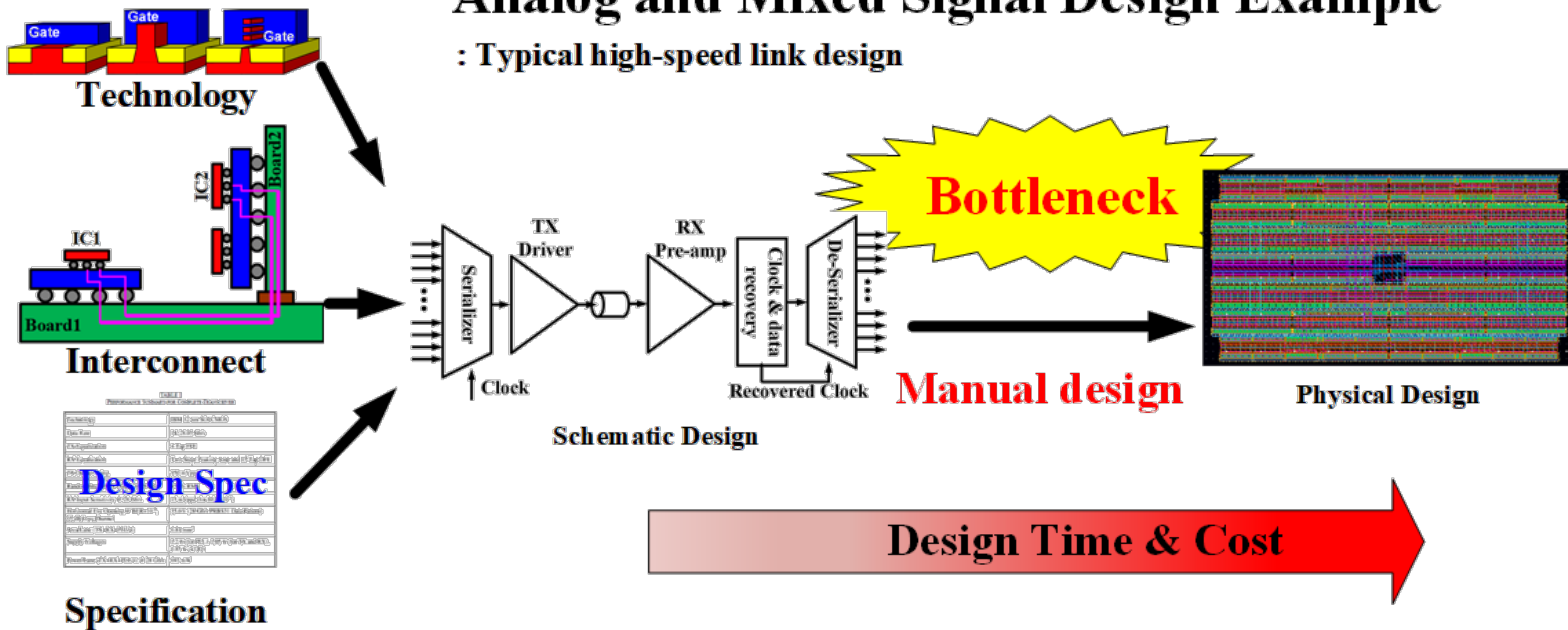
sgjeong@postech.ac.kr

<https://sungyu.dev>

The problems of analog layout automation

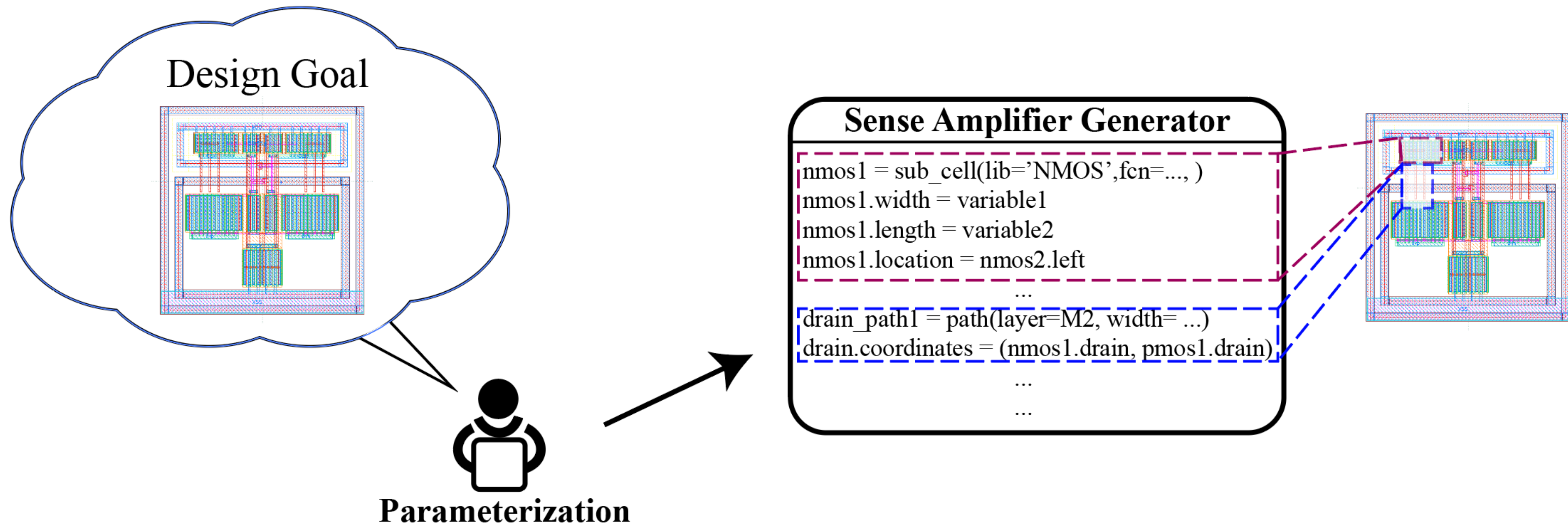
Analog and Mixed Signal Design Example

: Typical high-speed link design



■ Full-custom layout is the major bottleneck in circuit design.

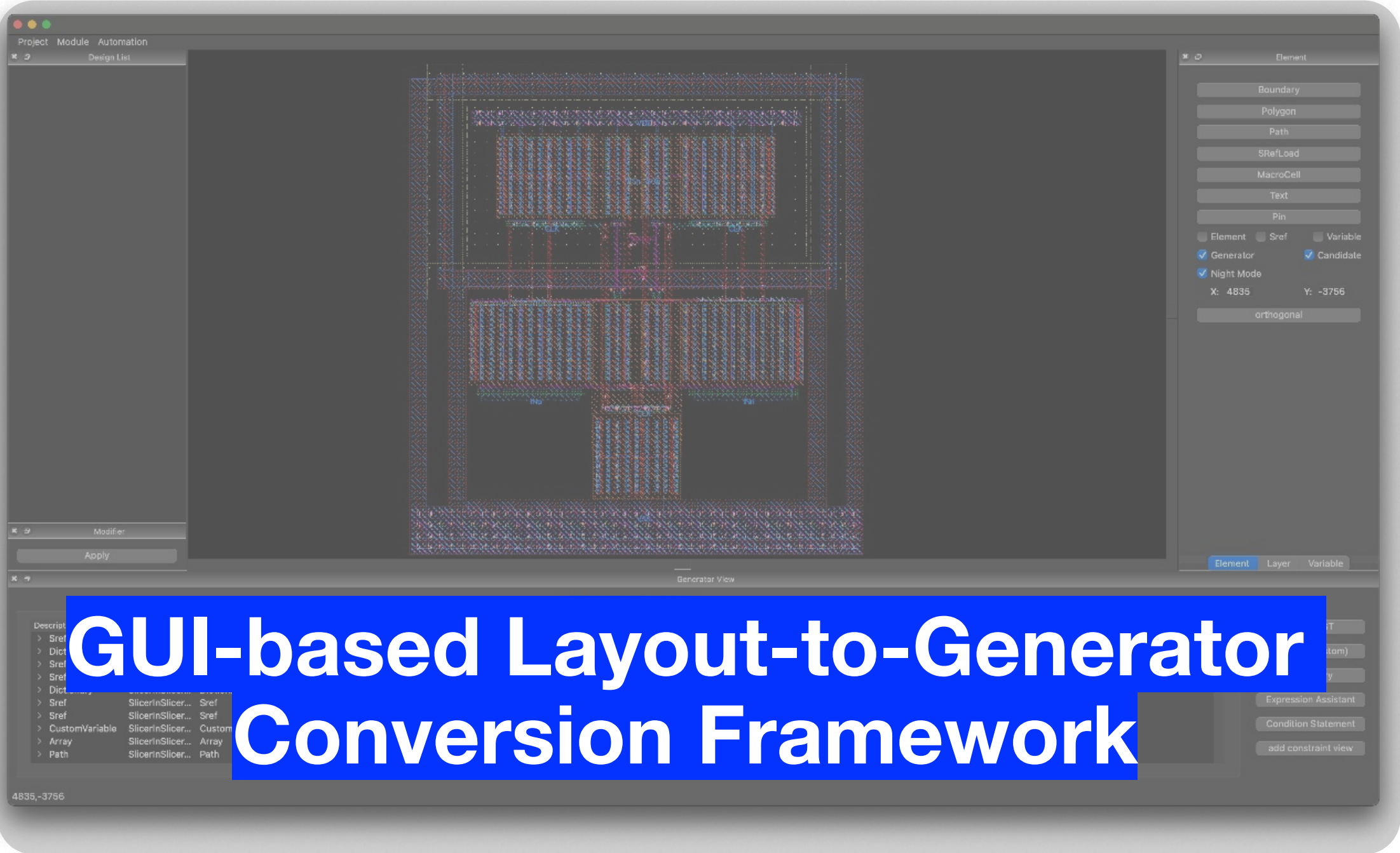
Previous Art: Procedural Layout Generator



- ☺ **Fast generation time**
- ☺ **Produces predictable output quality.**
- ☹ **Requires expertise in both analog circuits and programming.**
- ☹ **Requires significant work effort to develop generator.**

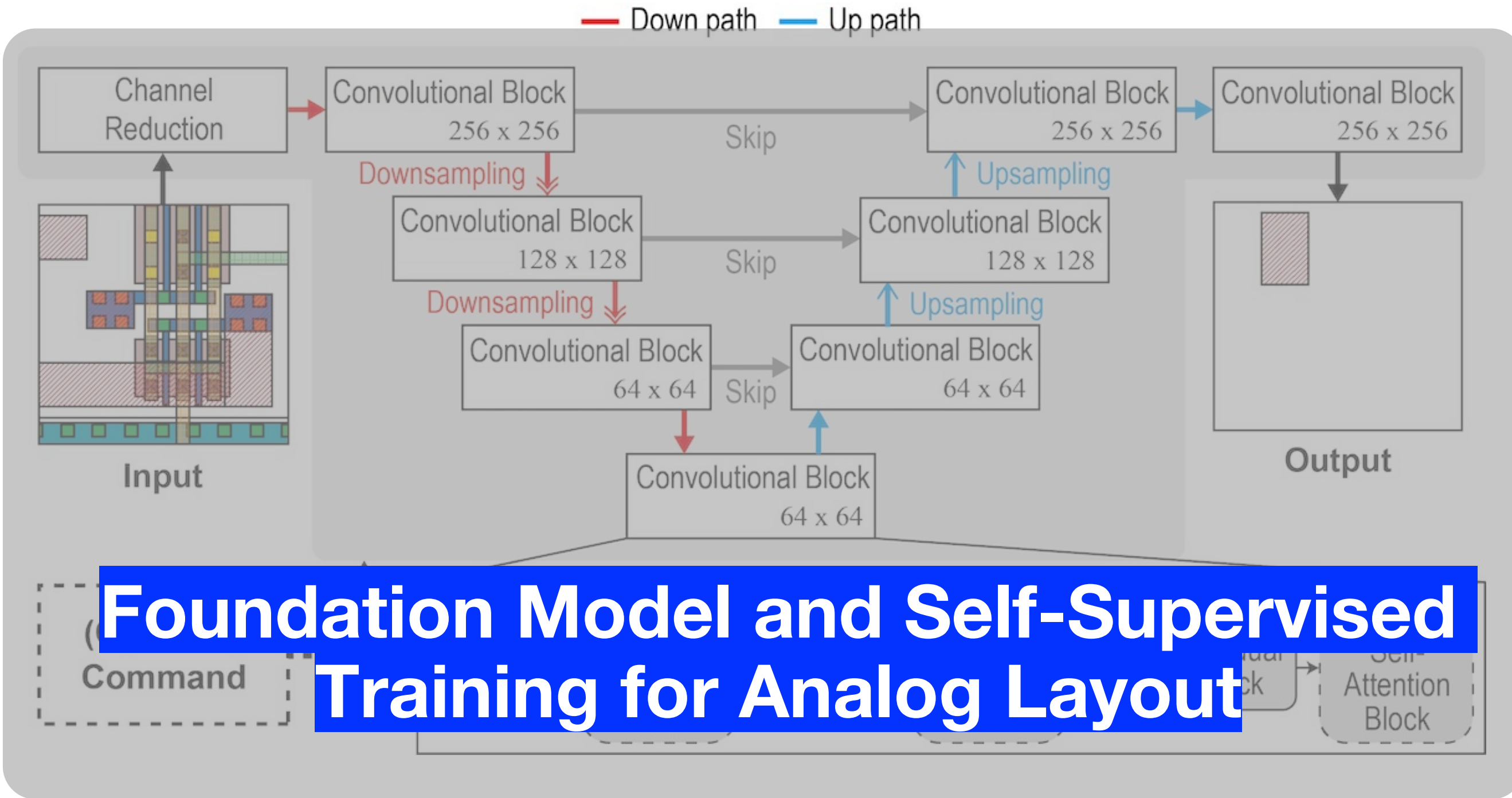
Suggested Solution: Reuse of Reference Designs

1) Explicit Reuse



- **Layout-to-generator conversion**
- **Deterministic approach**
- **Explicitly encodes design intent.**

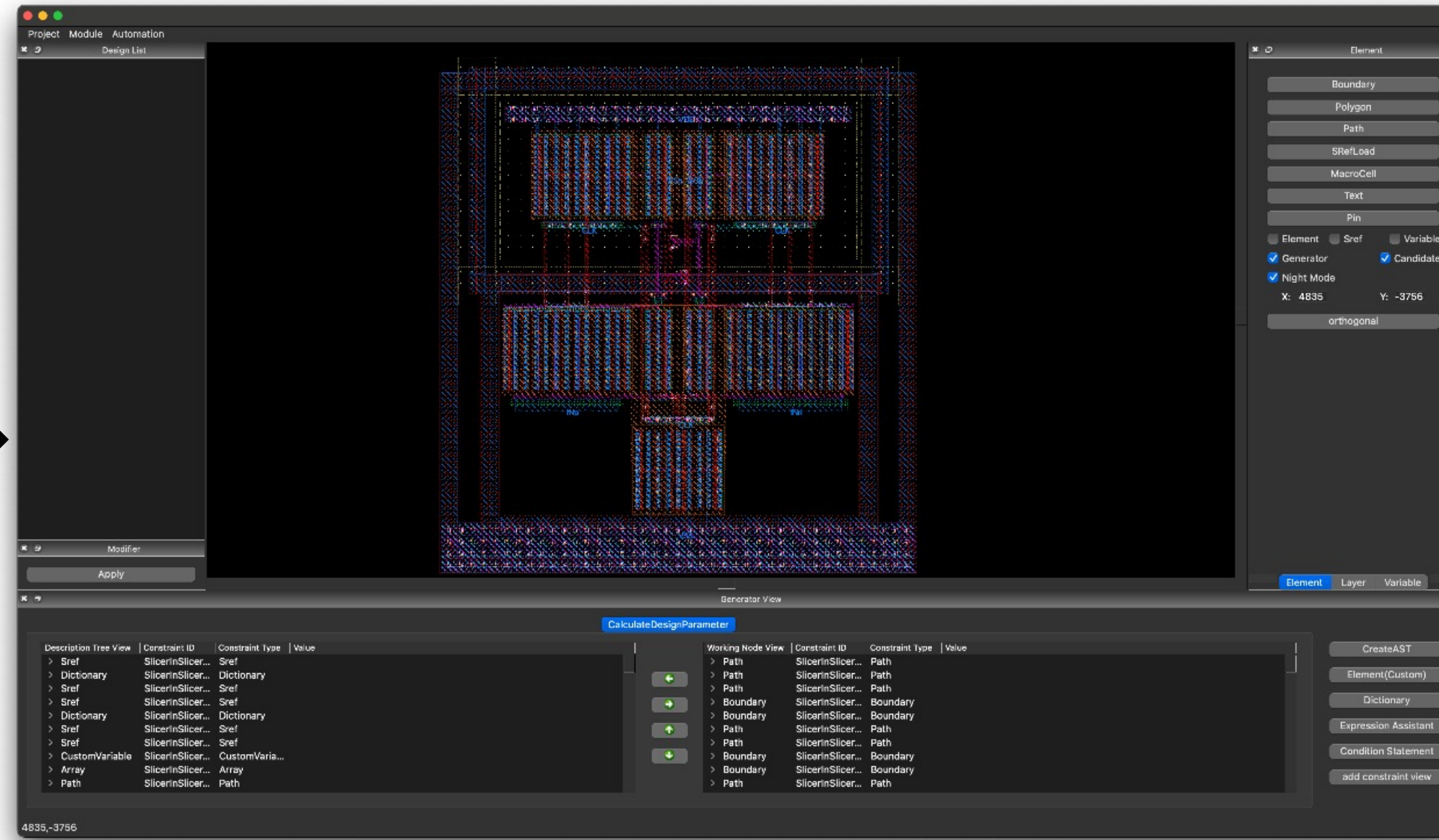
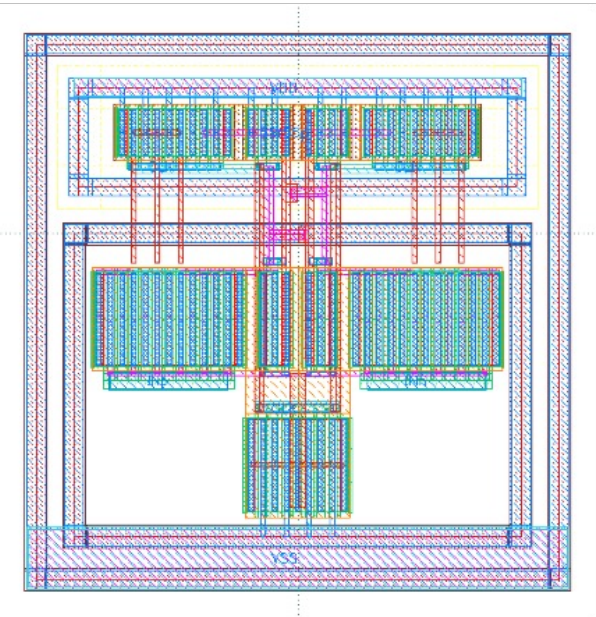
2) Implicit Reuse



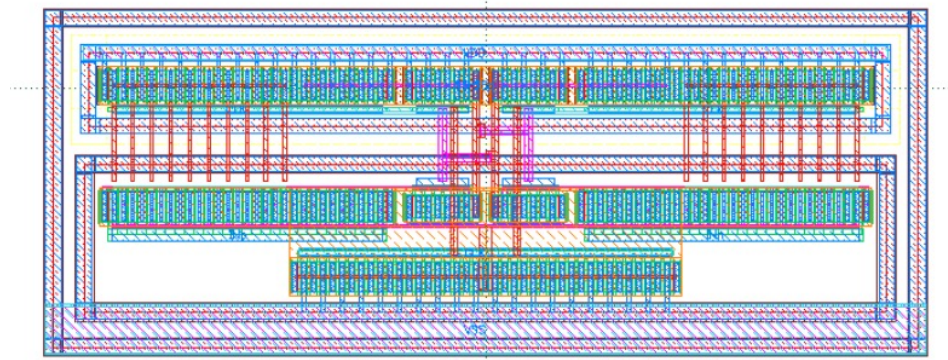
- **Self-supervised pre-training**
- **Probabilistic approach**
- **Implicitly learns design knowledge.**

Research 1) GUI-based Layout-to-Generator Conversion Framework

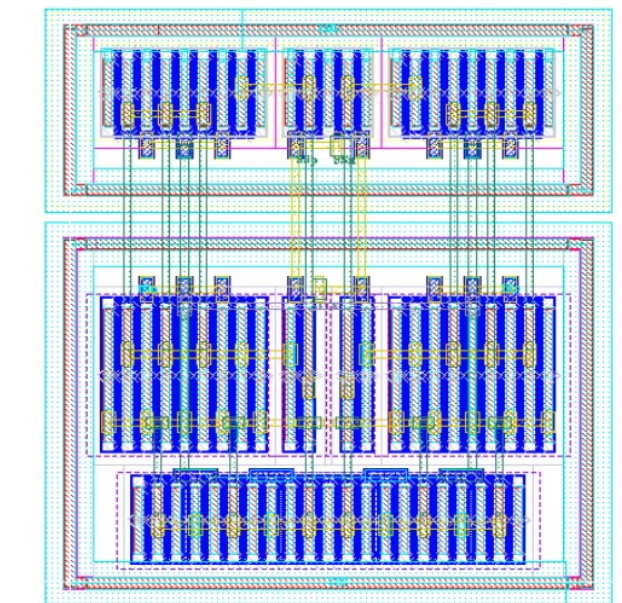
Fixed Layout



Layout Generator



Aspect ratio modification



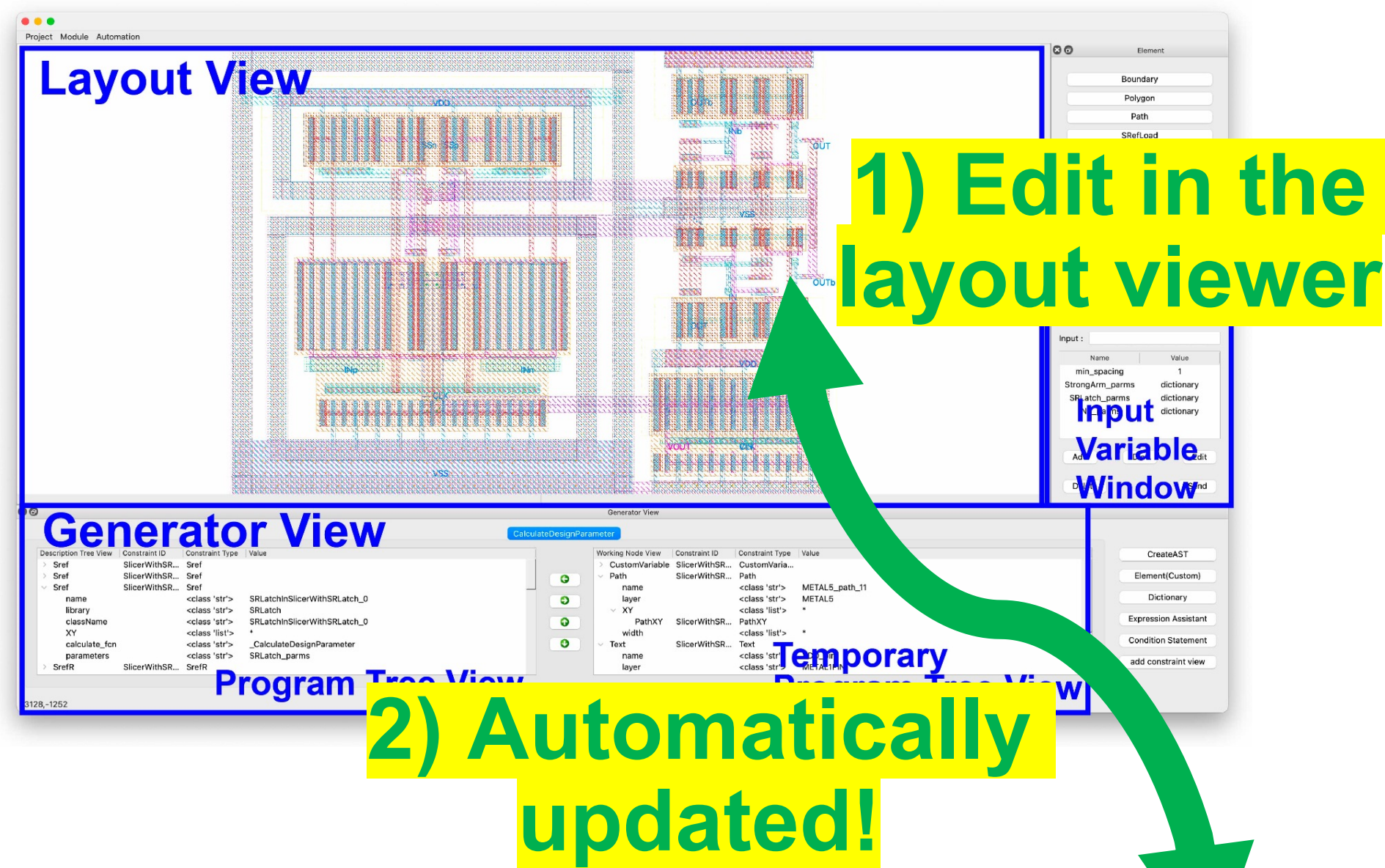
Technology porting



Proposed workflow: Load reference layout, build hierarchy, and describe parameterization expression **(whole process in GUI)**

- ☺ Requiring less knowledge of programming
- ☺ Leveraging well-made designs
- ☺ Reduced work effort and time

Research 1) GUI-based Layout-to-Generator Conversion Framework



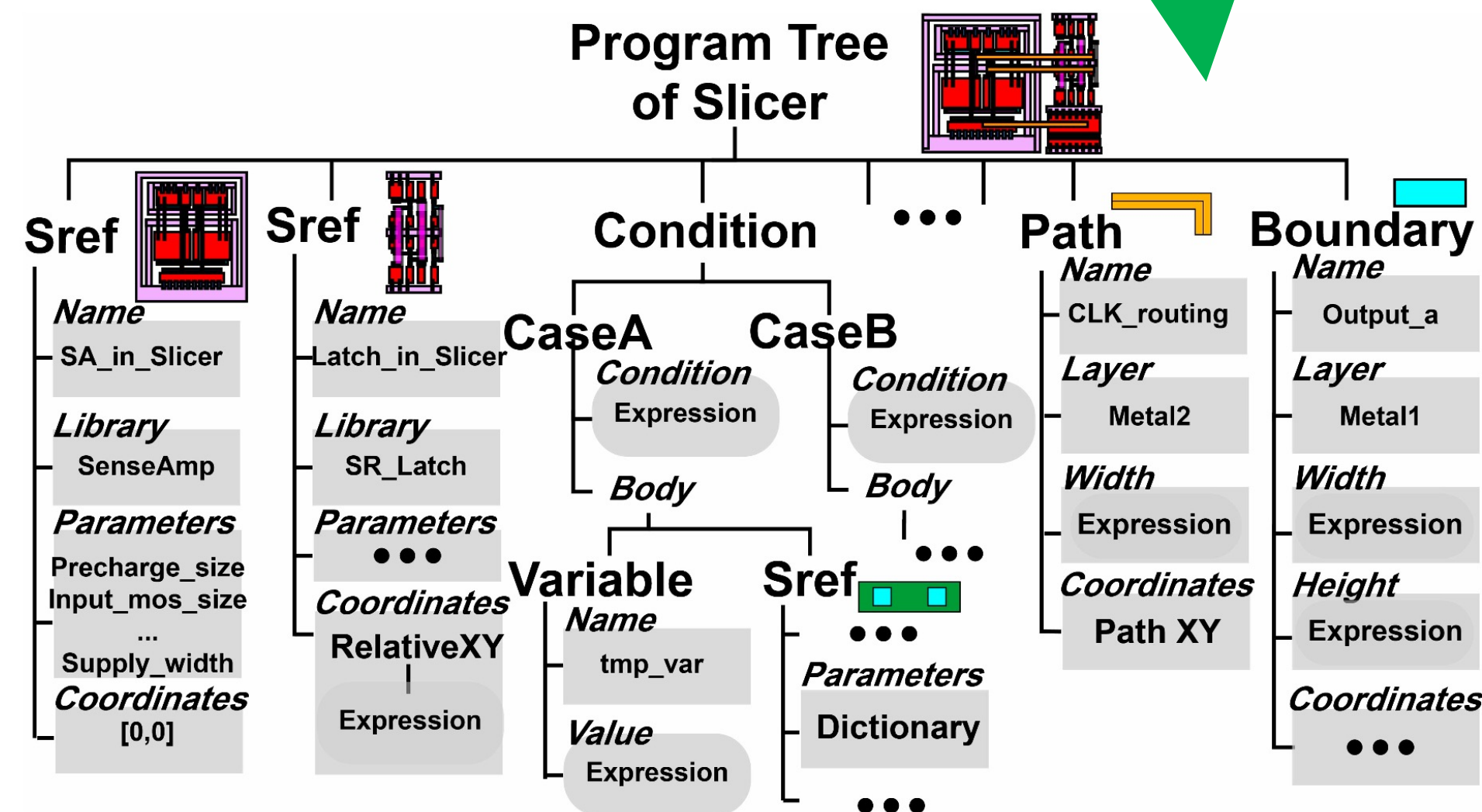
■ Users can edit directly in the layout viewer, similar to a conventional editor.

➔ 😊 Reduced workload

■ Visualized program block

■ Automatically compiled into Python scripts

➔ 😊 Programming becomes as easy as building Lego blocks.



3) Compilation

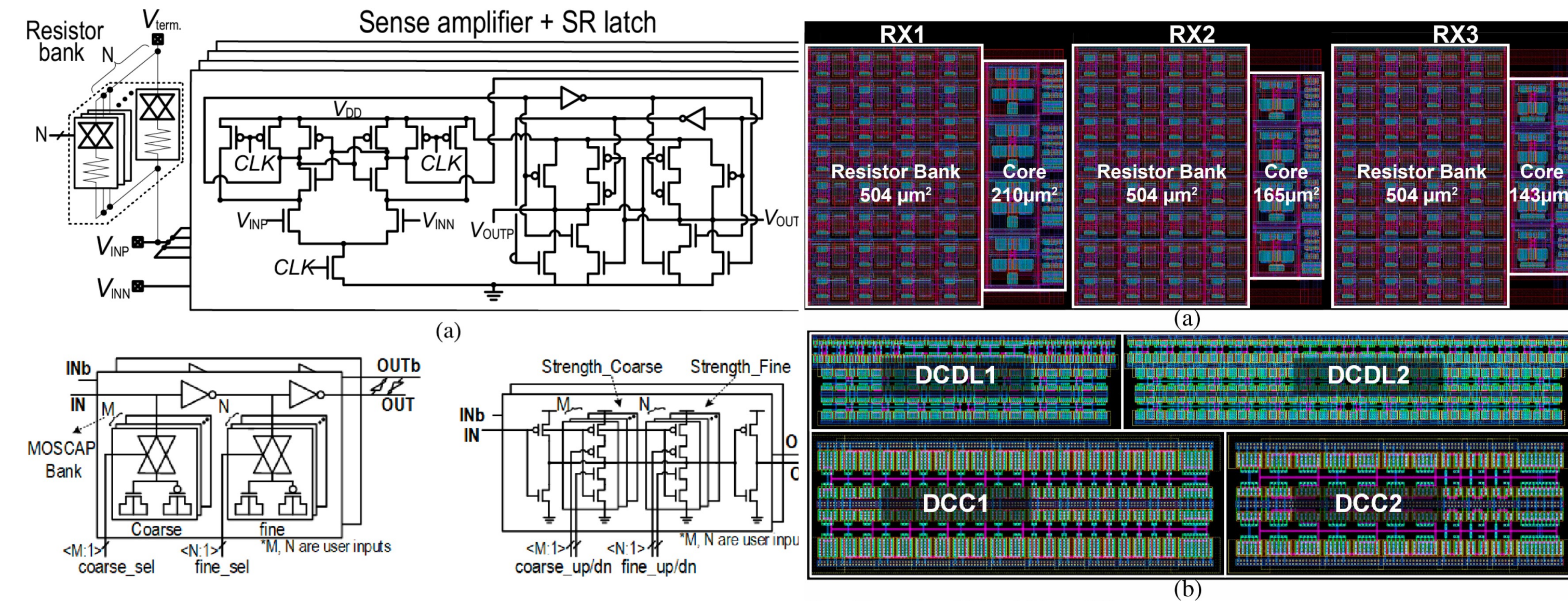


```

1 from generatorLib import DesignParameters
2 from generatorLib import DesignParameters
3 import math
4 from generatorLib import DesignParameters
5 from generatorLib.generator_models import StrongArmLatch
6 from generatorLib.generator_models import Inverter
7 from generatorLib.generator_models import ViaHole2D
8 from generatorLib.generator_models import ViaHole2D
9 from generatorLib.generator_models import ViaHole2D
10 from generatorLib.generator_models import SMLatch
11
12 class SMLatchGenerator(DesignParameters):
13     def __init__(self, DesignParameters, Name="SMLatchGenerator"):
14         super().__init__(DesignParameters, Name)
15         self._DesignParameter = DesignParameter
16
17     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
18         self._DesignParameter = DesignParameter
19
20     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
21         self._DesignParameter = DesignParameter
22
23     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
24         self._DesignParameter = DesignParameter
25
26     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
27         self._DesignParameter = DesignParameter
28
29     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
30         self._DesignParameter = DesignParameter
31
32     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
33         self._DesignParameter = DesignParameter
34
35     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
36         self._DesignParameter = DesignParameter
37
38     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
39         self._DesignParameter = DesignParameter
40
41     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
42         self._DesignParameter = DesignParameter
43
44     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
45         self._DesignParameter = DesignParameter
46
47     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
48         self._DesignParameter = DesignParameter
49
50     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
51         self._DesignParameter = DesignParameter
52
53     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
54         self._DesignParameter = DesignParameter
55
56     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
57         self._DesignParameter = DesignParameter
58
59     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
60         self._DesignParameter = DesignParameter
61
62     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
63         self._DesignParameter = DesignParameter
64
65     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
66         self._DesignParameter = DesignParameter
67
68     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
69         self._DesignParameter = DesignParameter
70
71     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
72         self._DesignParameter = DesignParameter
73
74     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
75         self._DesignParameter = DesignParameter
76
77     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
78         self._DesignParameter = DesignParameter
79
80     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
81         self._DesignParameter = DesignParameter
82
83     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
84         self._DesignParameter = DesignParameter
85
86     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
87         self._DesignParameter = DesignParameter
88
89     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
90         self._DesignParameter = DesignParameter
91
92     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
93         self._DesignParameter = DesignParameter
94
95     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
96         self._DesignParameter = DesignParameter
97
98     def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
99         self._DesignParameter = DesignParameter
100    def __DesignParameter__(self, Name="SMLatchGenerator", DesignParameter=DesignParameter):
    
```

Layout Generator

Research 1) Experimental Results



Cell	GUI (hour)	Script(hour)	Time Improvement
Receiver_top ¹	5	10.5	2.1x
Sense amplifier	5.5	13.5	2.5x
SR latch	4	11.5	2.9x
Resistor bank	5	7	1.4x
Inverter	0.5	2	4.0x
Receiver total	20	44.5	2.2x
DCDL_top ²	7.5	17	2.3x
MOSCAPbank unit	1.5	19.5	13.0x
Inverter	0.5	10	20.0x
DCDL total	9.5	46.5	4.9x
DCC_top ²	2.5	21	8.4x
Inverter	0.5	12	24.0x
Tri-state inverter	1.5	18	12.0x
DCC total	4.5	51	11.3x

Note
¹:Expert designers each GUI, Script
²:Novice designers each GUI, Script
 Reported time includes DRC+LVS runtime and code correction time.
 Time measurement number is rounded by half hour.

	RX1	RX2	RX3	
Maximum Data Rate (Gb/s)	20	16	12	
Active Region	Power (mW)	1.611	0.912	0.626
	Area (μm^2)	210	165	143
Resistor Bank	Power (mW)	8.487	8.548	8.601
	Area (μm^2)	504	504	504
Eye height (mV)	191.28	208.03	195.23	
Eye width (UI)	0.6324	0.6749	0.7513	
Generation time (sec)	0.289	0.289	0.284	

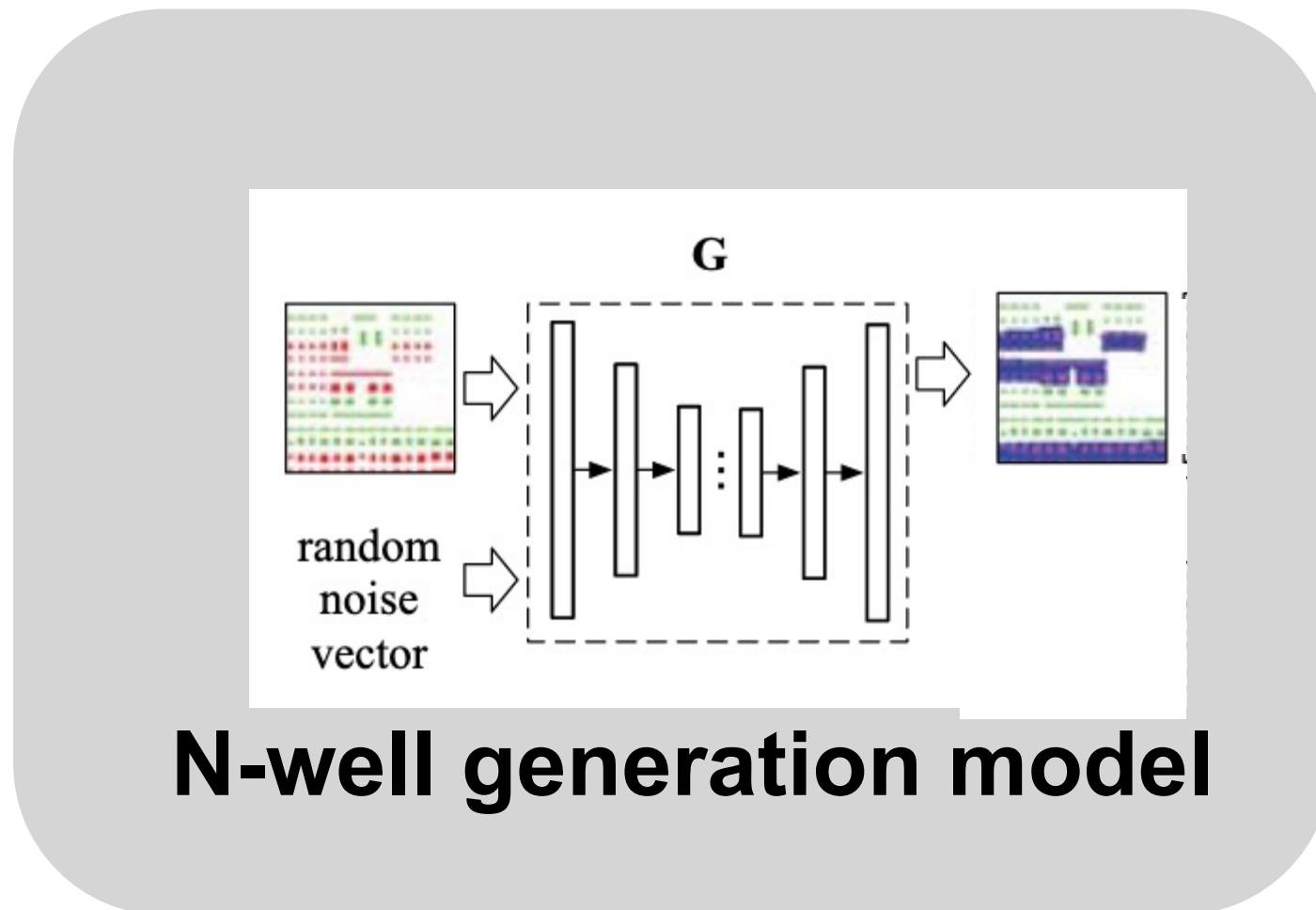
- **Converted three reference circuits into generators**
 (High-speed wireline receiver, digitally controlled delay line, and duty cycle corrector)
- **Generated more than hundreds instances**
- [Demo video link \(3:47\)](#)

Generates in < 0.3 sec

- Case study shows development time of procedural layout generator decreases dramatically.
(2.2X-11.3X faster)

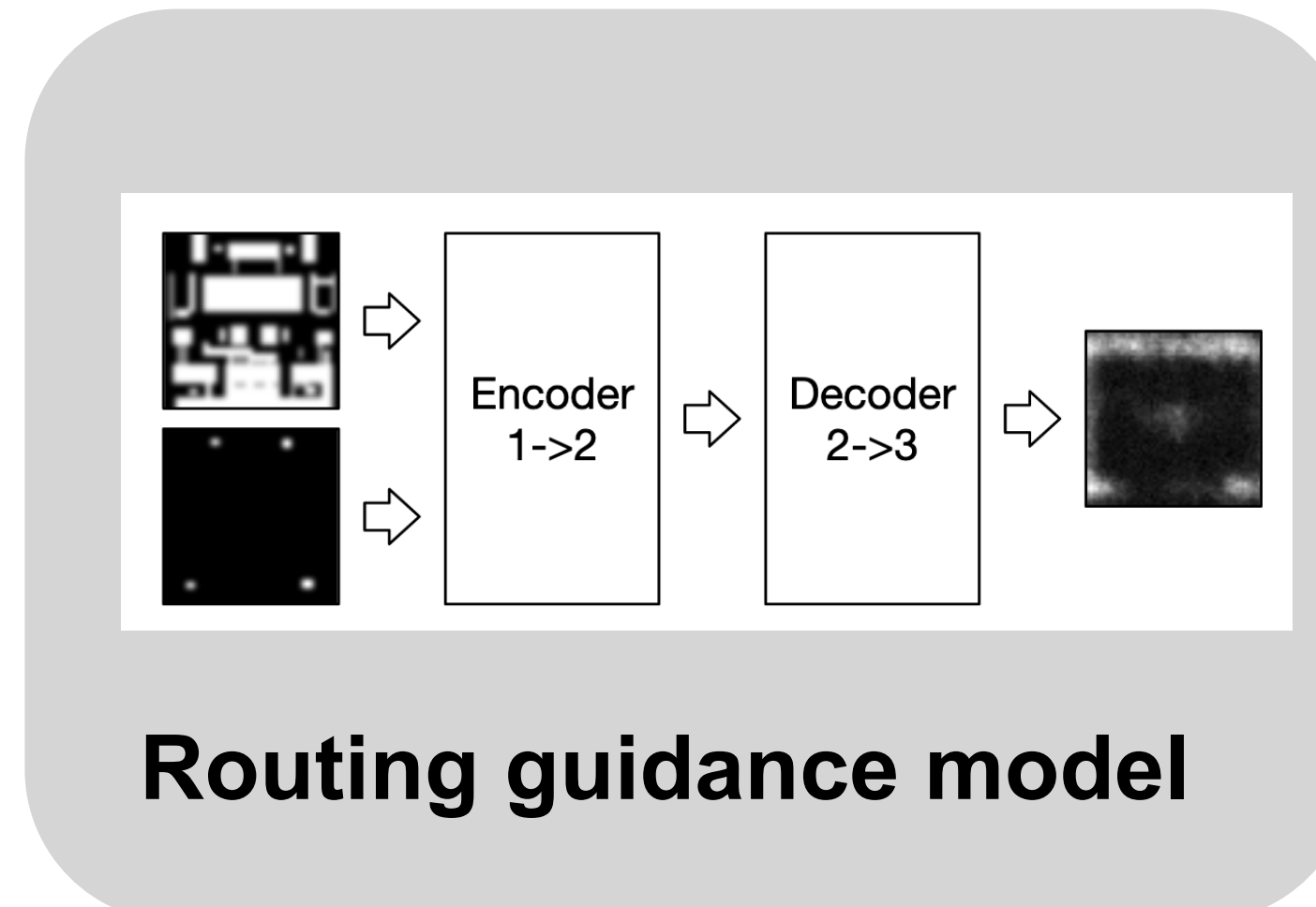
Research 2) Motivation

Task A



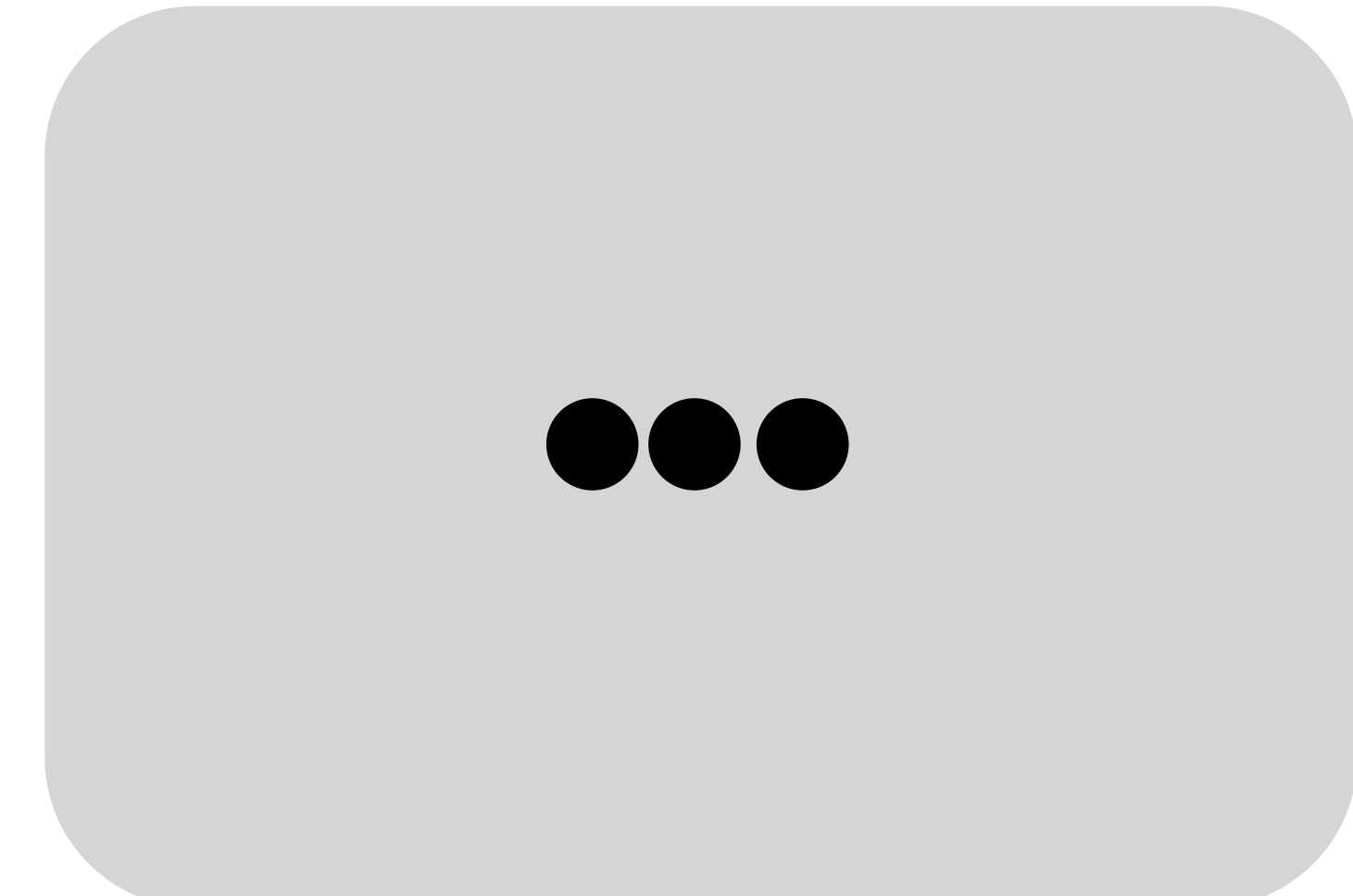
B. Xu *et al.*, "WellGAN: Generative-Adversarial-Network-Guided Well Generation for Analog/Mixed-Signal Circuit Layout," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, June 2019, pp. 1–6.

Task B



K. Zhu *et al.*, "GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Jan. 2019, pp. 1–8.

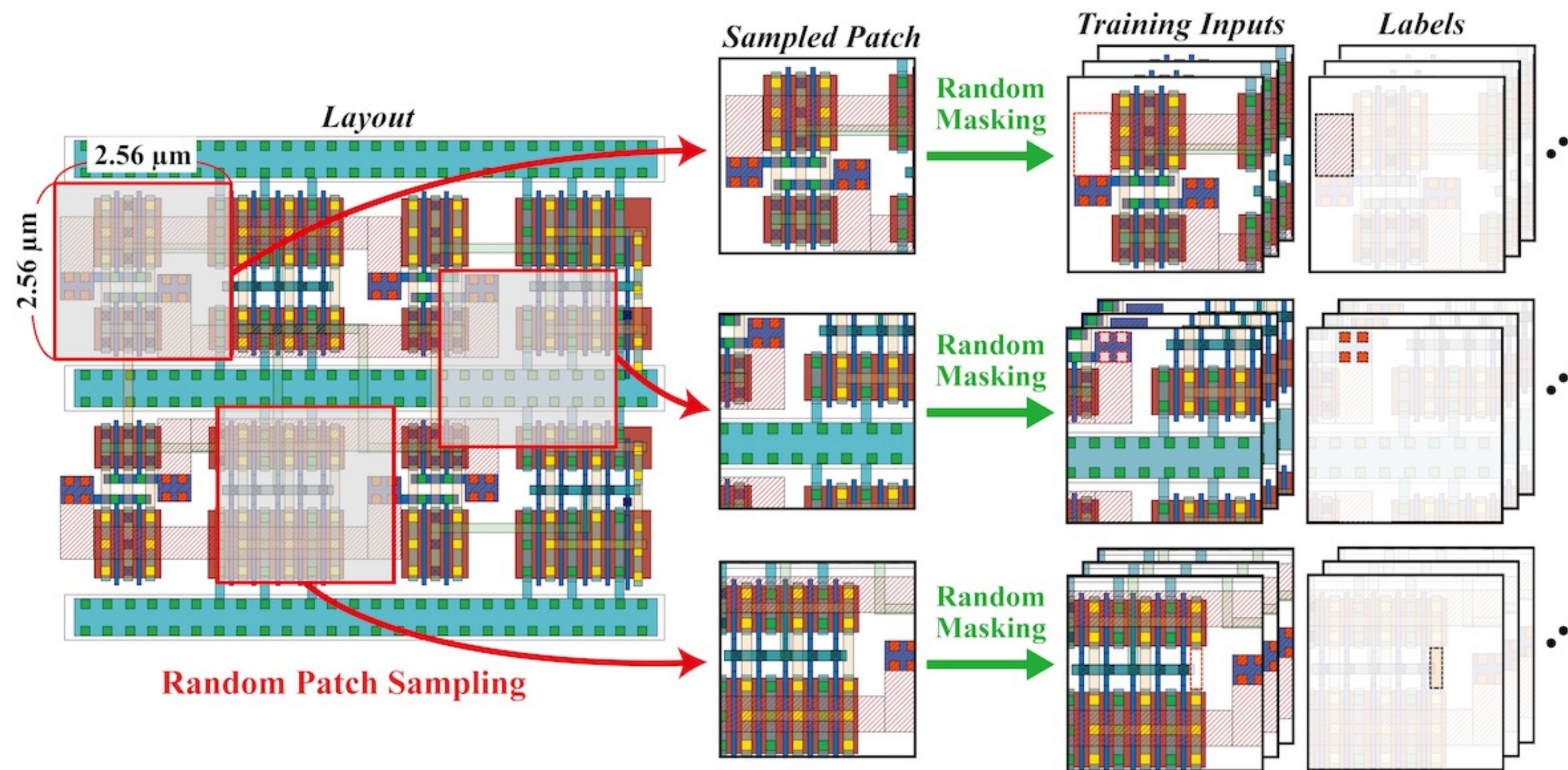
Task Z



Deep learning in analog layout is promising, but task-specific approaches face high labeling costs and development cost.

Research 2) Foundation Model & Self-Supervised Learning Method for Analog Layout Task

Self-supervised learning



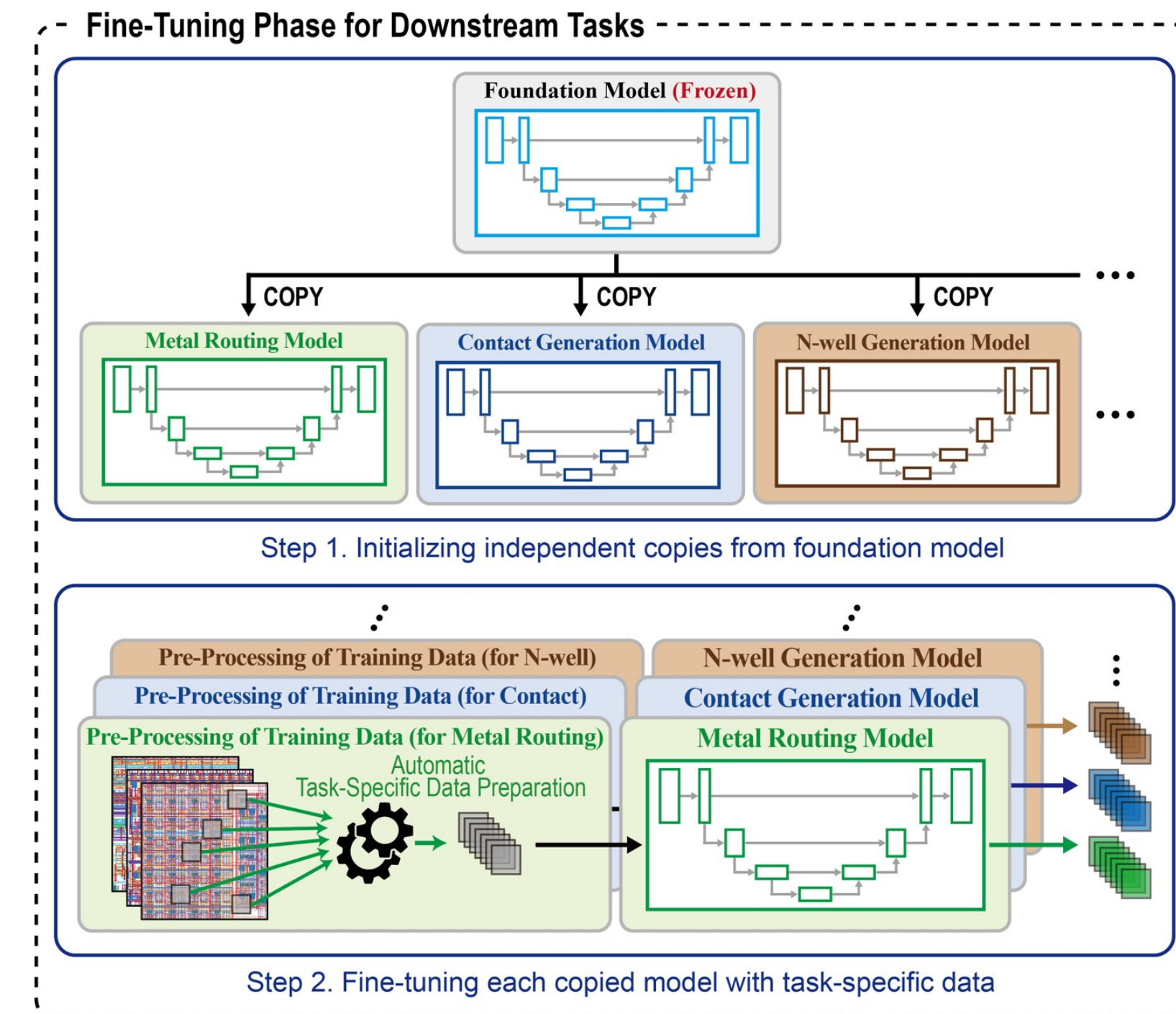
Input: Unlabeled data

Output: Trainable input/output pair

☺ **Leverages abundant unlabeled data** → **Low annotation cost**

☺ **Single foundation model for all tasks** → **High scalability**

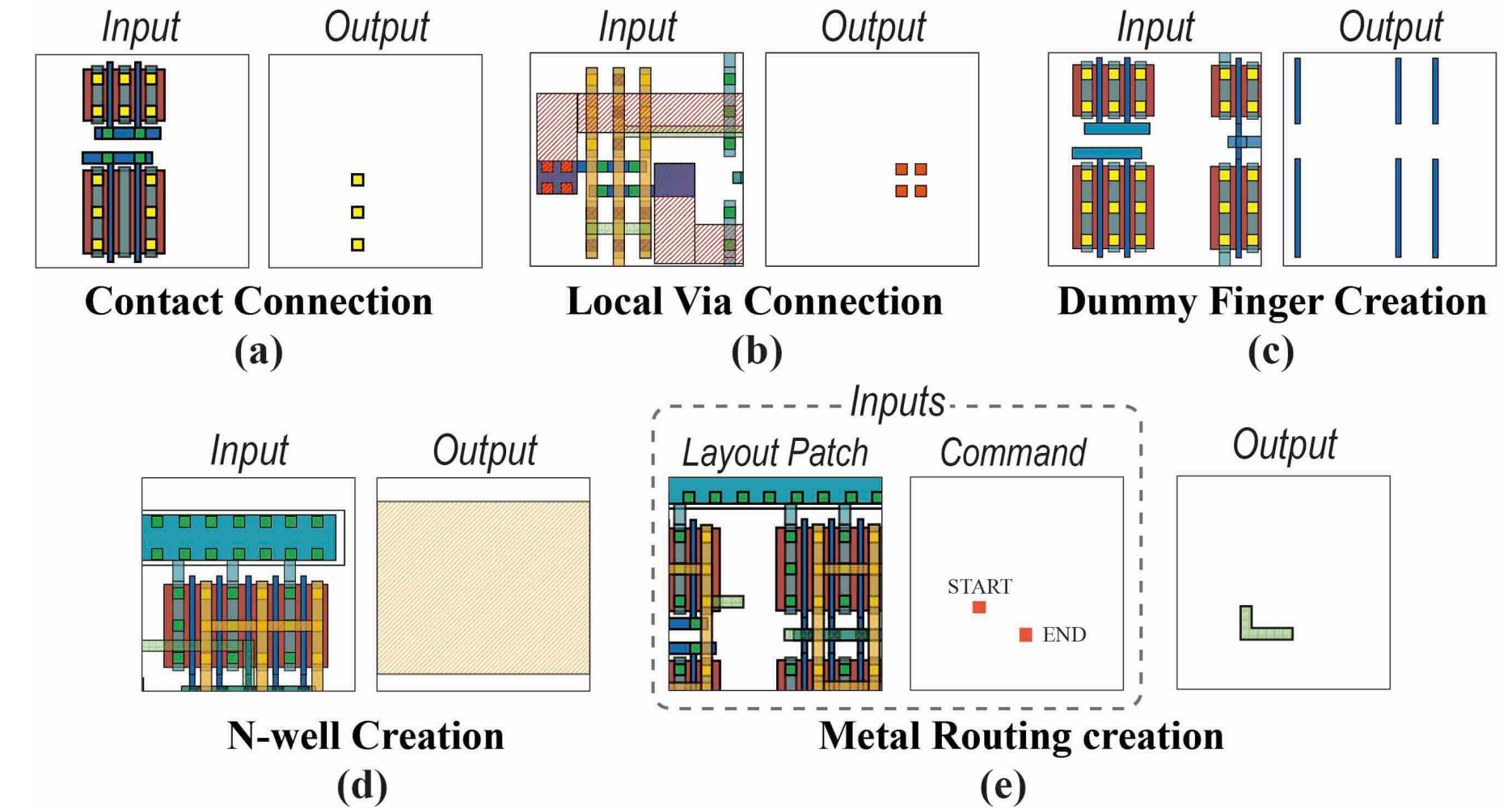
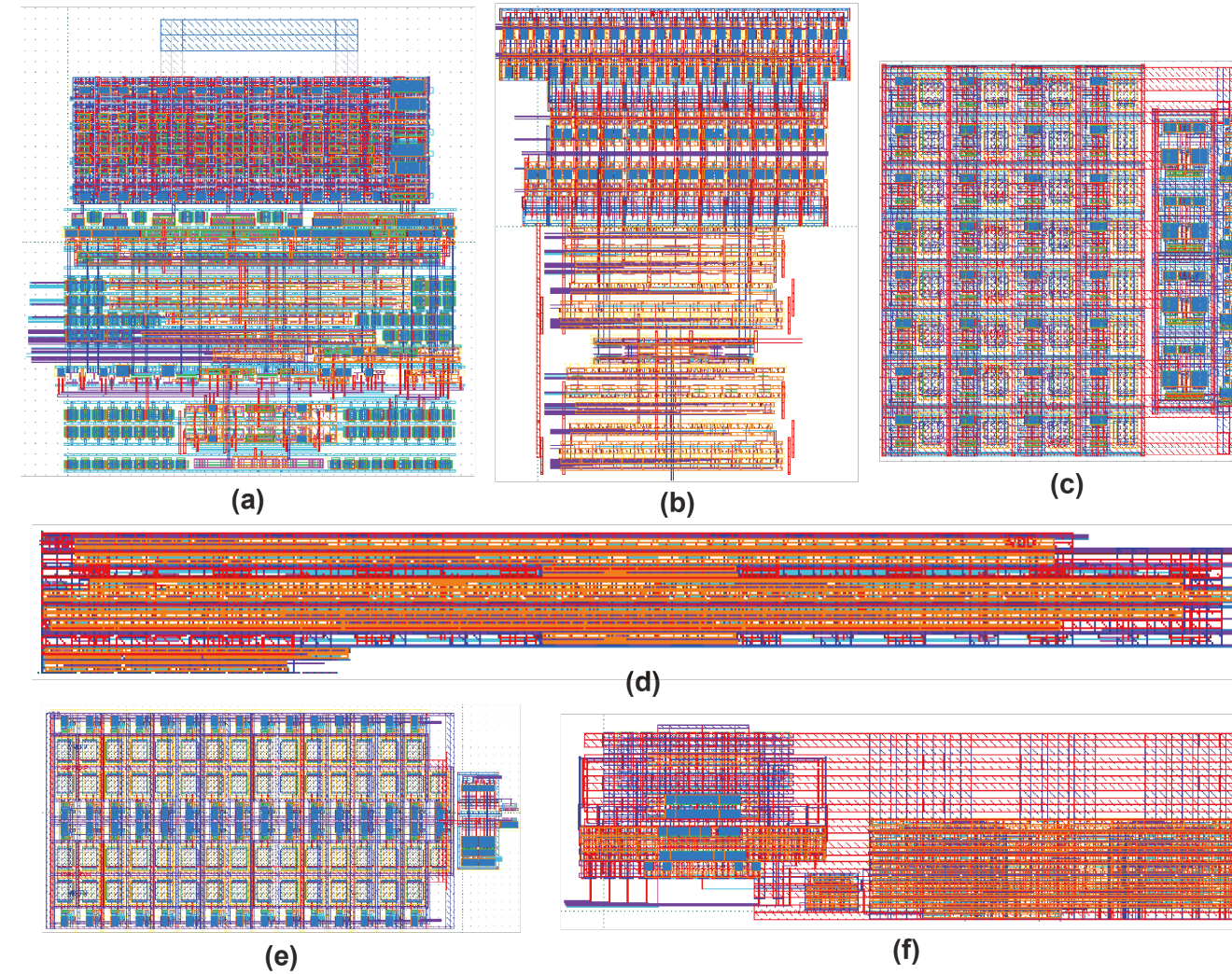
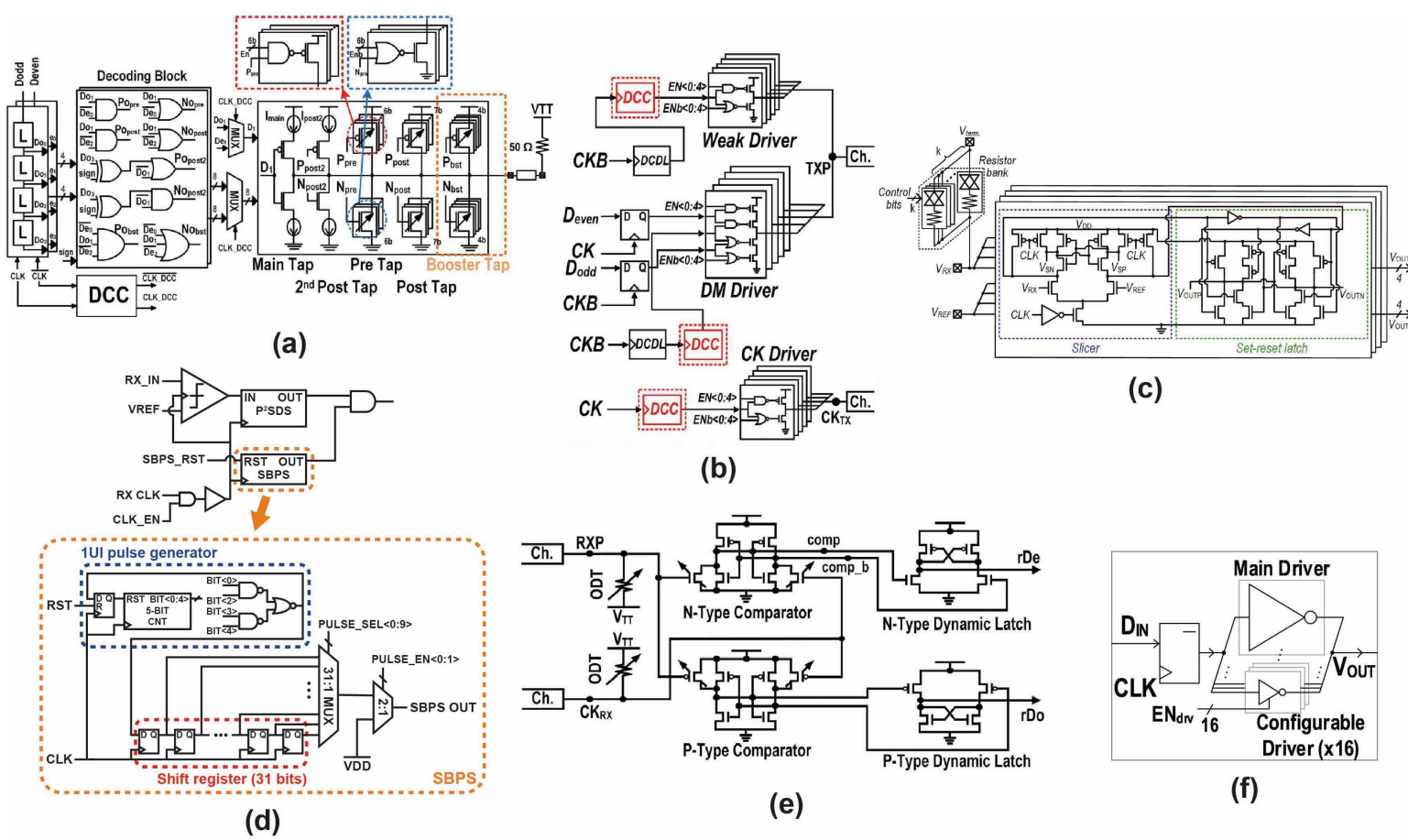
Foundation model



Task-agnostic foundation model

➔ Fine-tuning for diverse downstream tasks

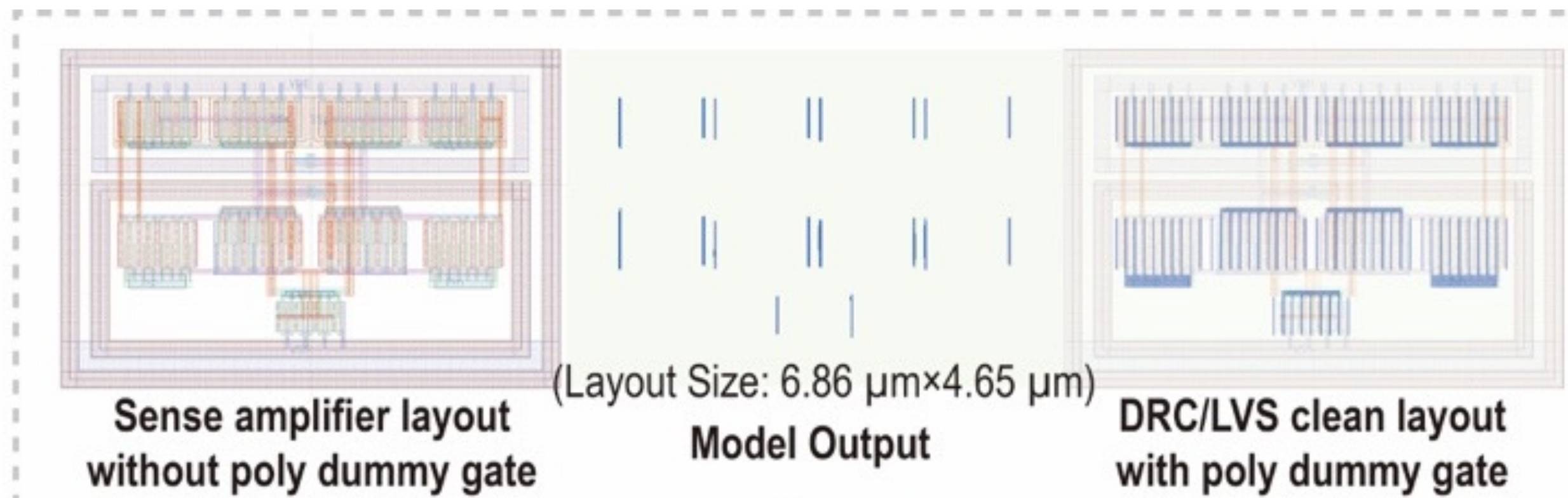
Research 2) Experimental Results



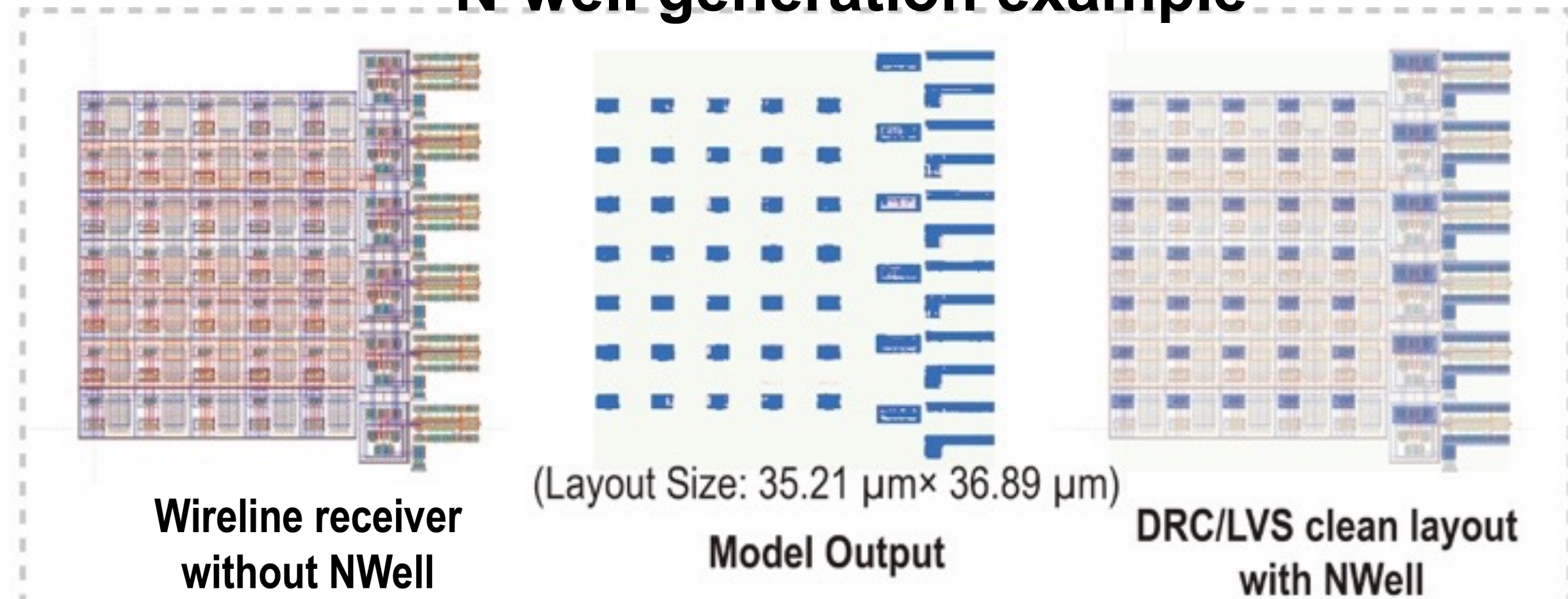
- Self-supervised pre-training on 6 different system-level AMS circuits
- 6 circuits → **324,000** training samples

- Fine-tuning for five different downstream tasks

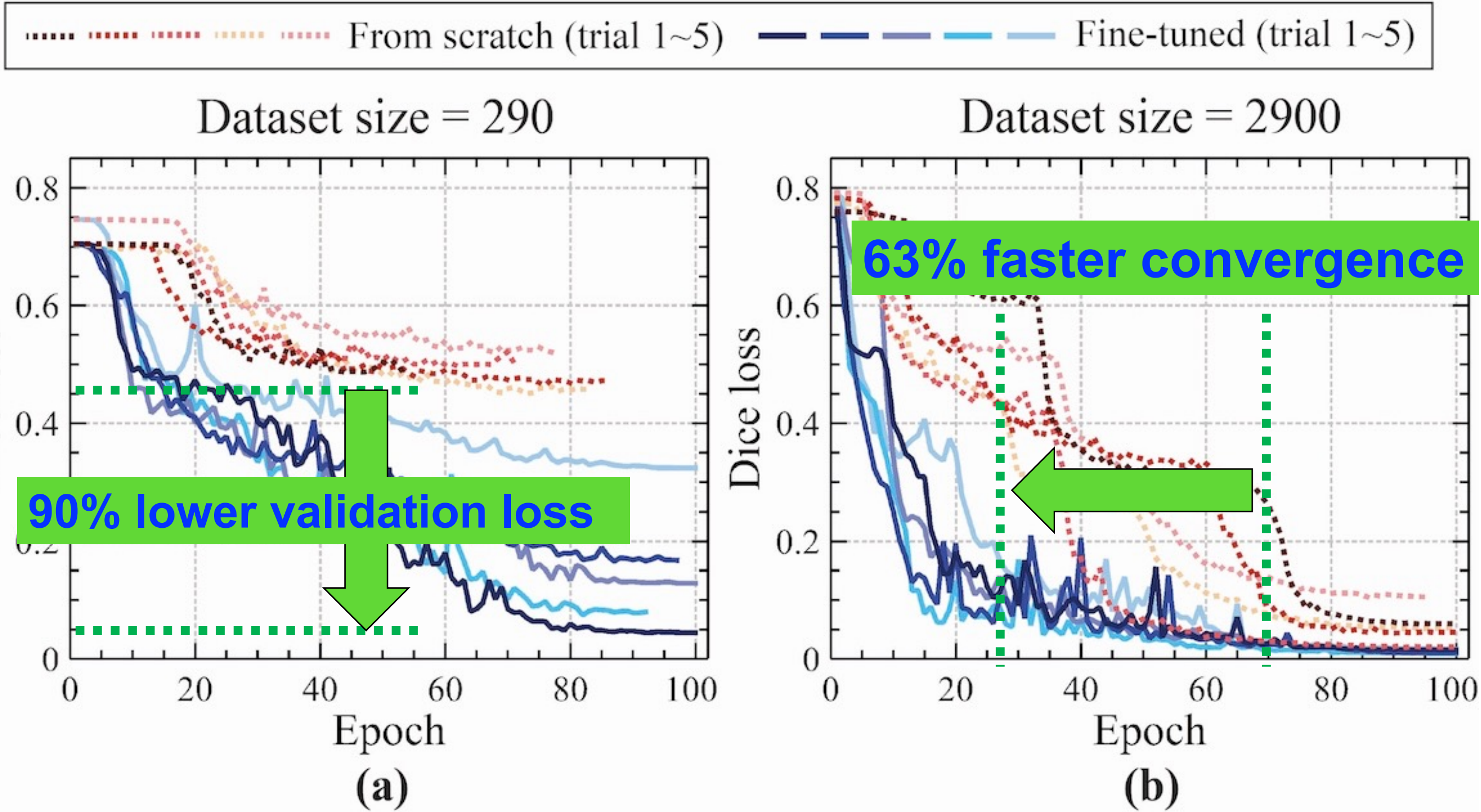
Dummy finger generation example



N-well generation example



Research 2) Experimental Results



Comparison with prior works

Input Context	Ground Truth	Ours	WellGAN

Input Context	Ground Truth	Ours	GeniusRoute

- 😊 **Fine-tuning: Fast convergence & lower validation loss**
- ☹️ **Train-from-scratch: Unstable convergence & overfitting**

Models achieved **96.6%** DRC/LVS-clean success on **1,824** benchmark samples across five tasks.