Neural Network Hardware Acceleration and LLM-Driven Circuit Design

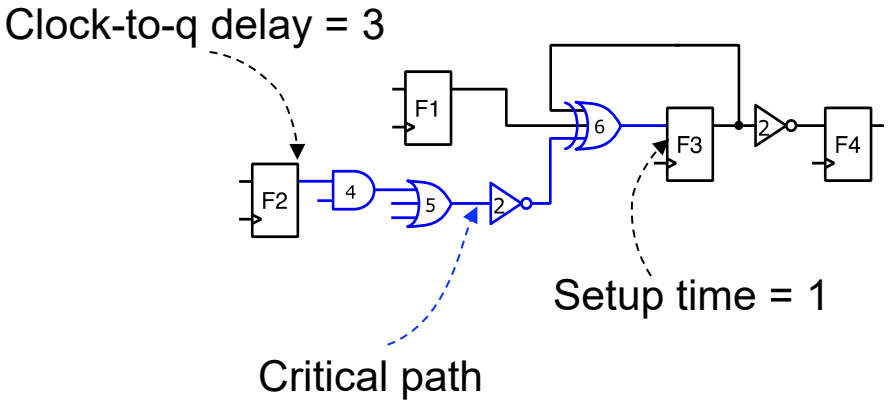
Bing Li

FG Ressourceneffiziente Künstliche Intelligenz (ReKI)

Digital Circuits

- **Circuit optimization**
- Circuit camouflage

Timing of Digital Circuits

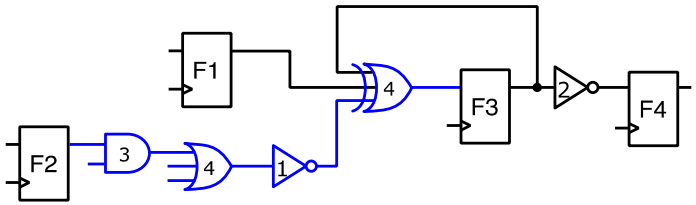


T_{min}=3+(4+5+2+6)+1=21 units

- Circuit performance determined by the longest path (critical path)
- Minimum clock period calculated by summing the clock-to-q delay, critical path delay and the setup time

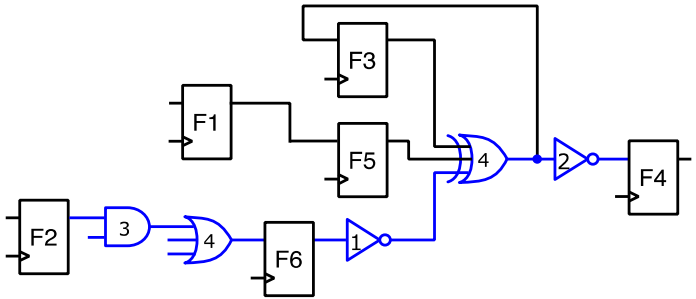
Timing Optimization Techniques

Gate Sizing



$T_{min}=3+12+1=16$

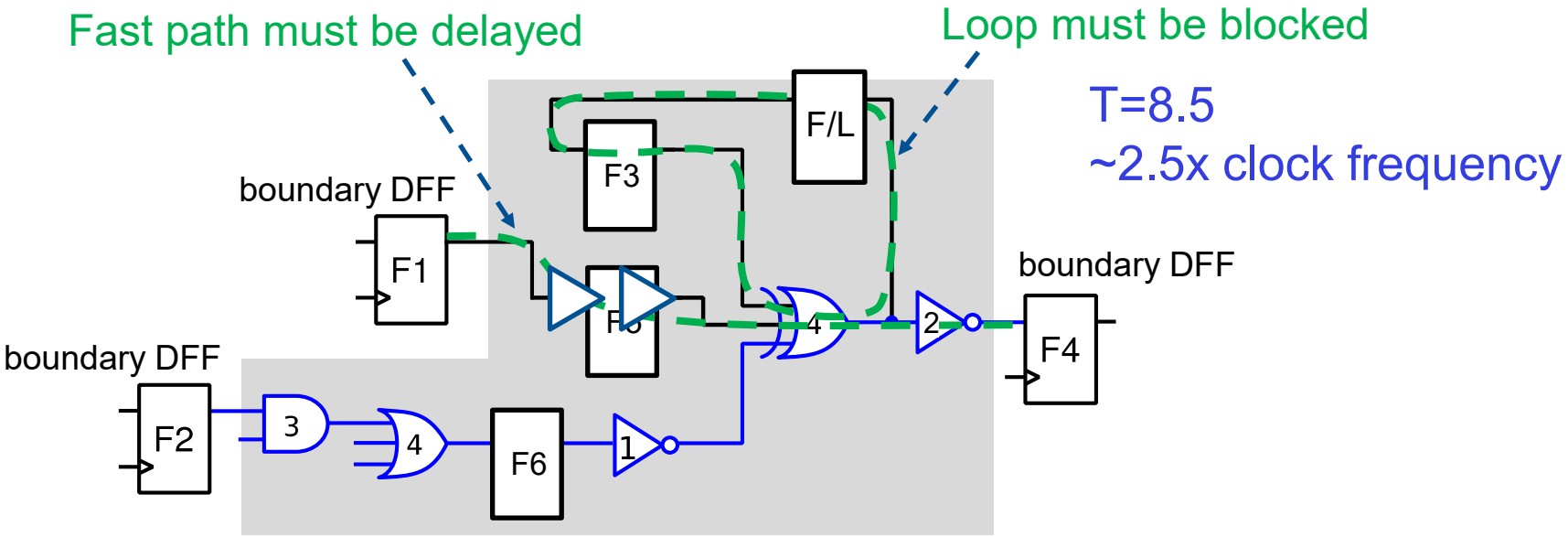
Retiming



$T_{min}=3+7+1=11$

The limit in the traditional timing paradigm

VirtualSync



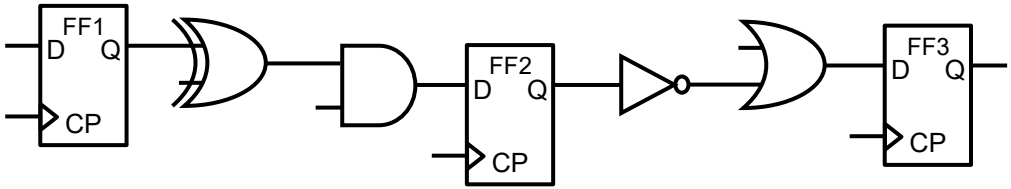
- Remove flip-flops inside the selected subcircuit
- Size gates on long paths
- Block fast signals on short paths and loops for timing synchronization

Li Zhang, Bing Li, Xing Huang, Xunzhao Yin, Cheng Zhuo, Masanori Hashimoto, Ulf Schlichtmann, “VirtualSync+: Timing Optimization with Virtual Synchronization”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), December 2022

Digital Circuits

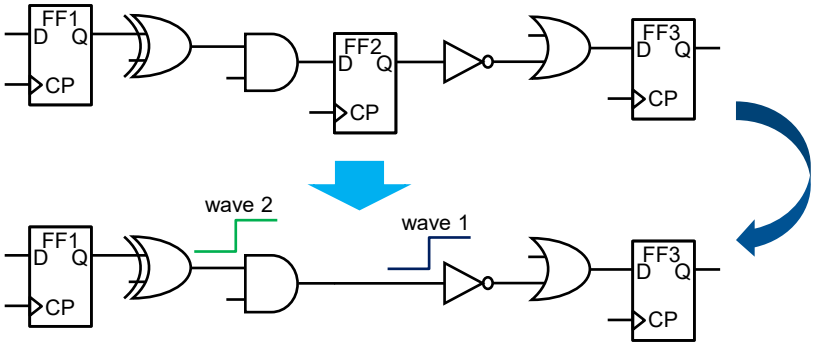
- Circuit optimization
- **Circuit camouflage**

Counterfeiting with Conventional Timing

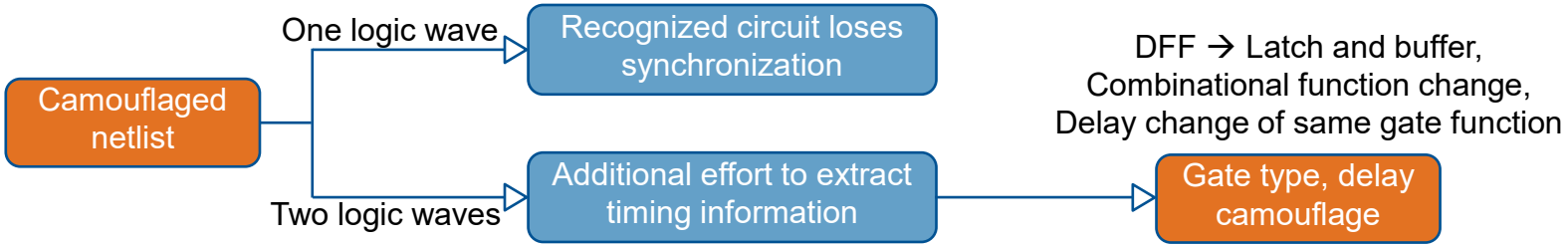


- Conventional timing model
  - All paths defined with respect to one clock period
  - Setup and hold time constraints satisfied between pairs of flip-flops
- A netlist is sufficient to reproduce a circuit using a standard EDA flow.
- State of the art of anti-counterfeiting: logic locking

Anti-Counterfeiting with Wave Pipelining



Wave pipelining paths shown as single-cycle paths in reverse engineering



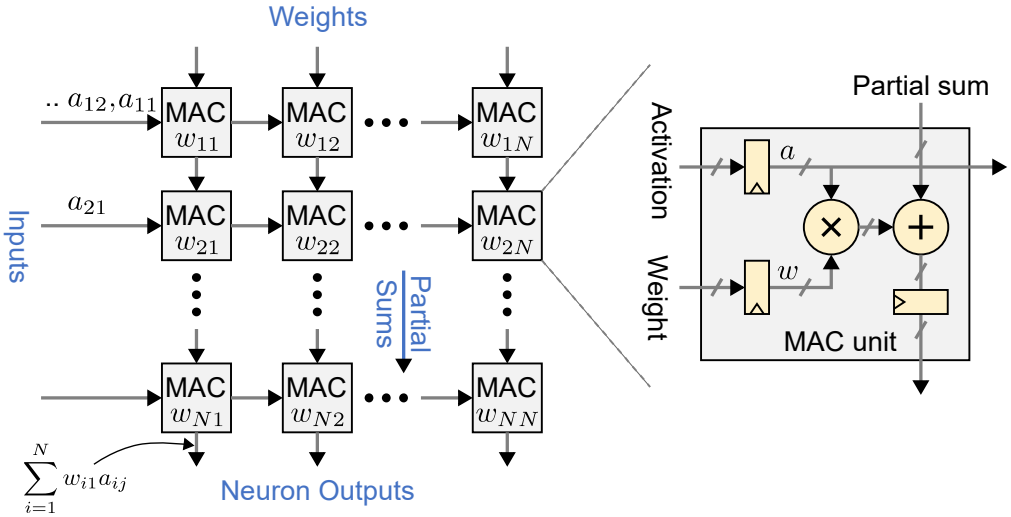
DFF → Latch and buffer,
Combinational function change,
Delay change of same gate function

Li Zhang, Bing Li, Meng Li, Bei Yu, David Z. Pan, Michaela Brunner, Georg Sigl, Ulf Schlichtmann, "TimingCamouflage+: Netlist Security Enhancement with Unconventional Timing", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, December 2020

Digital Neural Network Acceleration

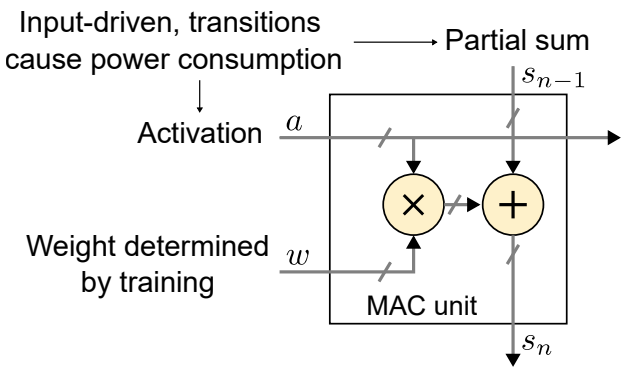
- **Low-power computing with weight selection**
- Circuit reduction with encoding
- Circuit simplification with fixed weights

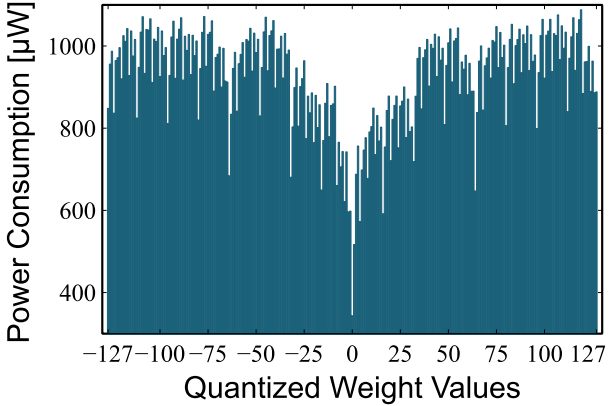
Digital Hardware Acceleration of Neural Networks



- Weights and inputs pipelined through a systolic array for a better performance
- MAC operations implemented in digital circuits → need to balance PPA

Power-driven Weight Selection

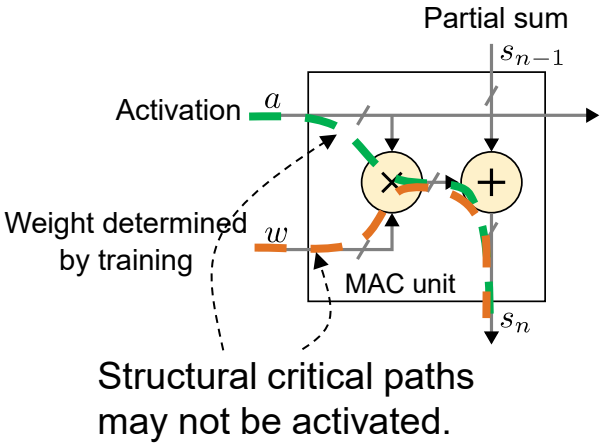


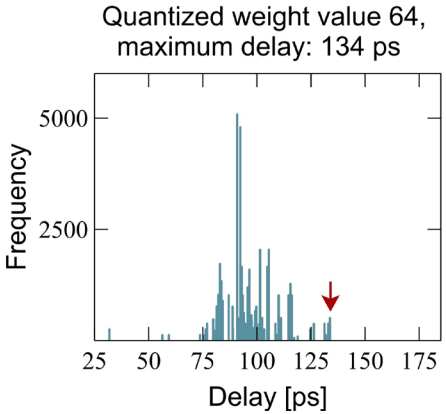


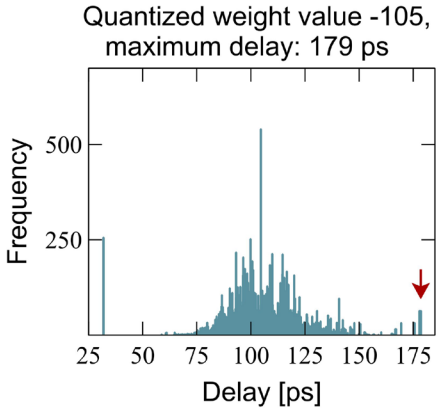
- Power consumption of a MAC unit determined by input transitions
- Select weight values according to average power consumption

N. P. Jouppi et al., “In-datacenter performance analysis of a tensor processing unit,” Int. Symp. Comput. Arch. (ISCA), 2017.

Delay-driven Weight Selection

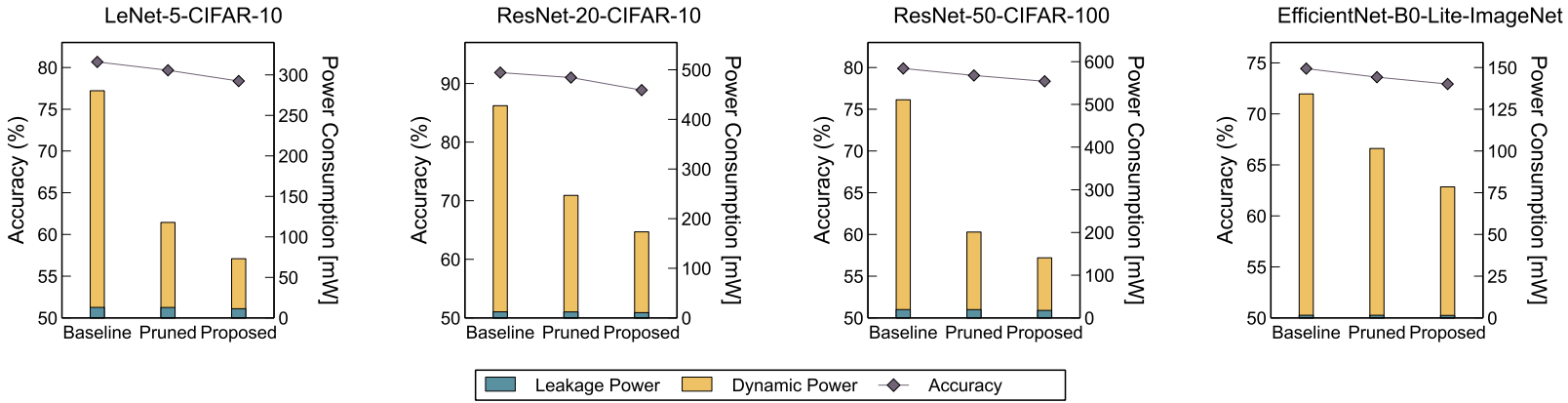






- Select weight values and activations according to the delays of circuit paths triggered in the MAC units
- Voltage scaling V↓ to reduce power consumption $P \sim V^2$

Power Efficiency Enhancement with PowerPruning



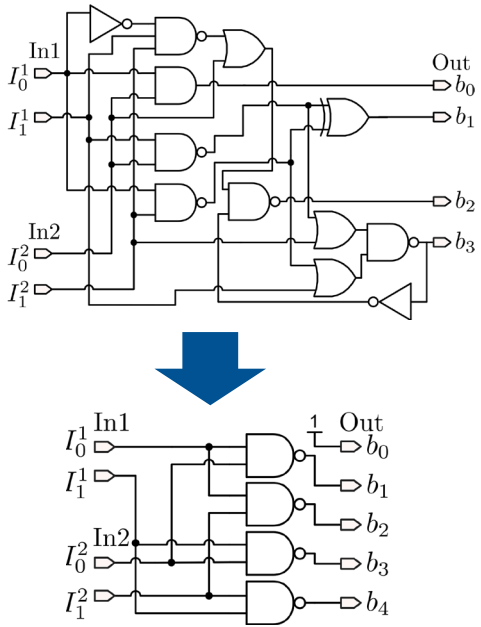
- Power consumption reduced by up to 38% compared with pruning
- No hardware modification in MAC units

R. Petri, L. Zhang, Y. Chen, U. Schlichtmann, B. Li, “PowerPruning: Selecting Weights and Activations for Power-Efficient Neural Network Acceleration”, *ACM/IEEE Des. Autom. Conf. (DAC),* 2023

Digital Neural Network Acceleration

- Low-power computing with weight selection
- **Circuit reduction with encoding**
- Circuit simplification with fixed weights

Multiplier Simplification with Encoding



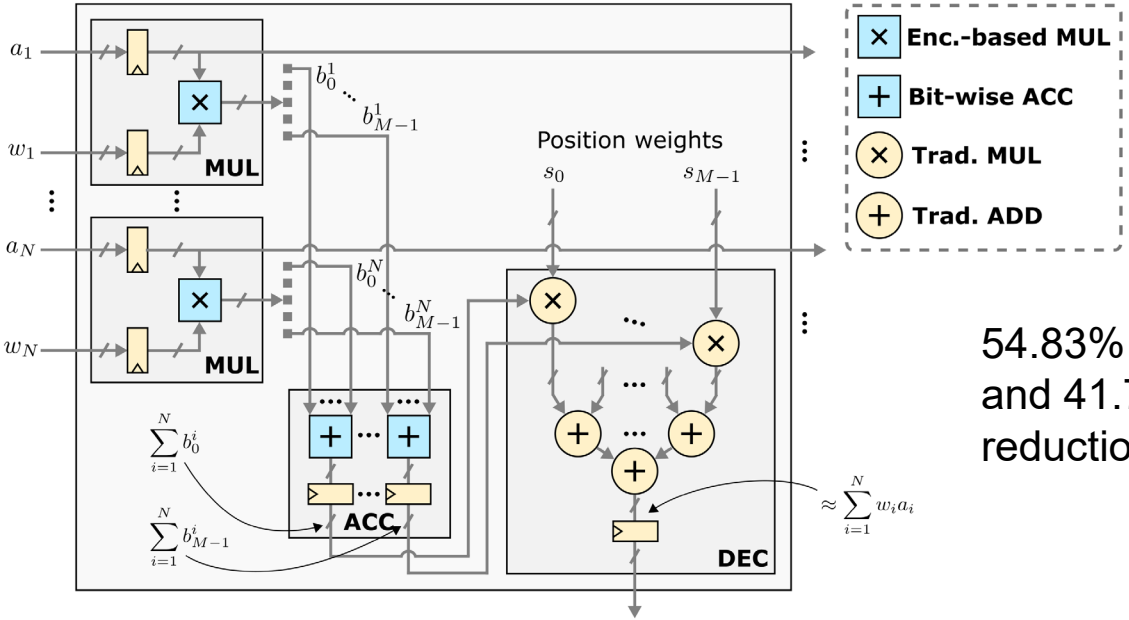
Multiplication simplification by new encoding

In1 $I_1^1 I_0^1$	In2 $I_1^2 I_0^2$	Trad. Enc. $b_3 b_2 b_1 b_0$	New Enc. $b_4 b_3 b_2 b_1 b_0$	Value v
10	10	0100	01111	4
10	11	0010	00111	2
10	00	0000	11111	0
10	01	1110	10111	-2
11	10	0010	01011	2
11	11	0001	00001	1
11	00	0000	11111	0
11	01	1111	10101	-1
00	10	0000	11111	0
00	11	0000	11111	0
00	00	0000	11111	0
00	01	0000	11111	0
01	10	1110	11011	-2
01	11	1111	11001	-1
01	00	0000	11111	0
01	01	0001	11101	1

$v = \sum_{i=0}^{M-1} s_i b_i$, where s_i is position weight.
Trad.: $M = 4, s_3 = -8, s_2 = 4, s_1 = 2, s_0 = 1$
New: $M = 5, s_4 = -4, s_3 = 2, s_2 = 2, s_1 = -1, s_0 = 1$

B. Liu, B. Li, L. Zhang, X. Yin, C. Zhuo, U. Schlichtmann, “EncodingNet: A Novel Encoding-based MAC Design for Efficient Neural Network Acceleration”, *IEEE Transactions on Circuits and Systems for Artificial Intelligence, 1(2),* 2024

Array-Level Architecture Optimization with Encoding



54.83% area reduction and 41.74% power reduction in average

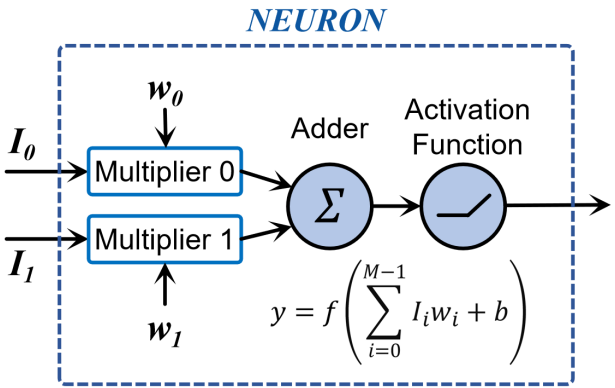
MAC array with bit-wise accumulator (ACC) and a decoder (DEC)

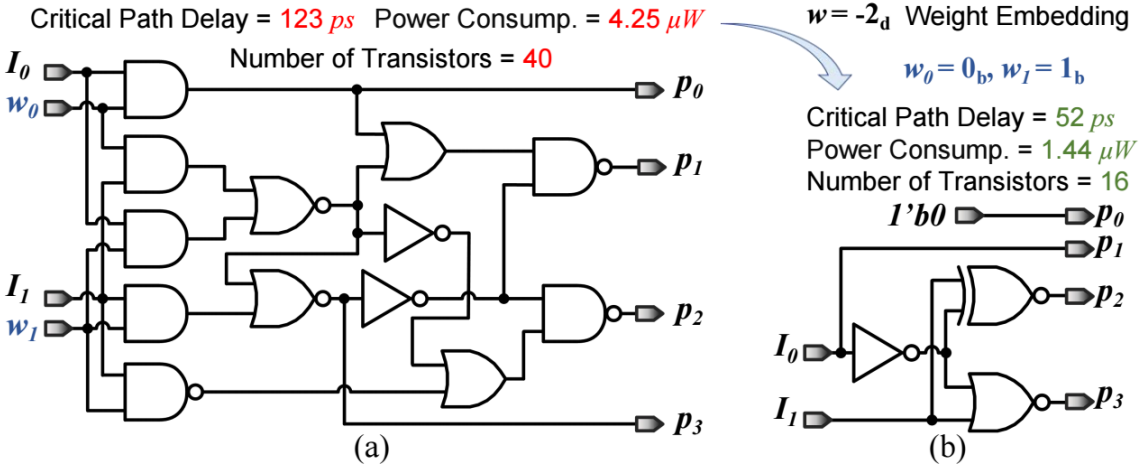
B. Liu, B. Li, L. Zhang, X. Yin, C. Zhuo, U. Schlichtmann, “EncodingNet: A Novel Encoding-based MAC Design for Efficient Neural Network Acceleration”, *IEEE Transactions on Circuits and Systems for Artificial Intelligence,* 1(2), Dec. 2024

Digital Neural Network Acceleration

- Low-power computing with weight selection
- Circuit reduction with encoding
- **Circuit flattening with fixed weights**

Logic Implementation: Multiplier

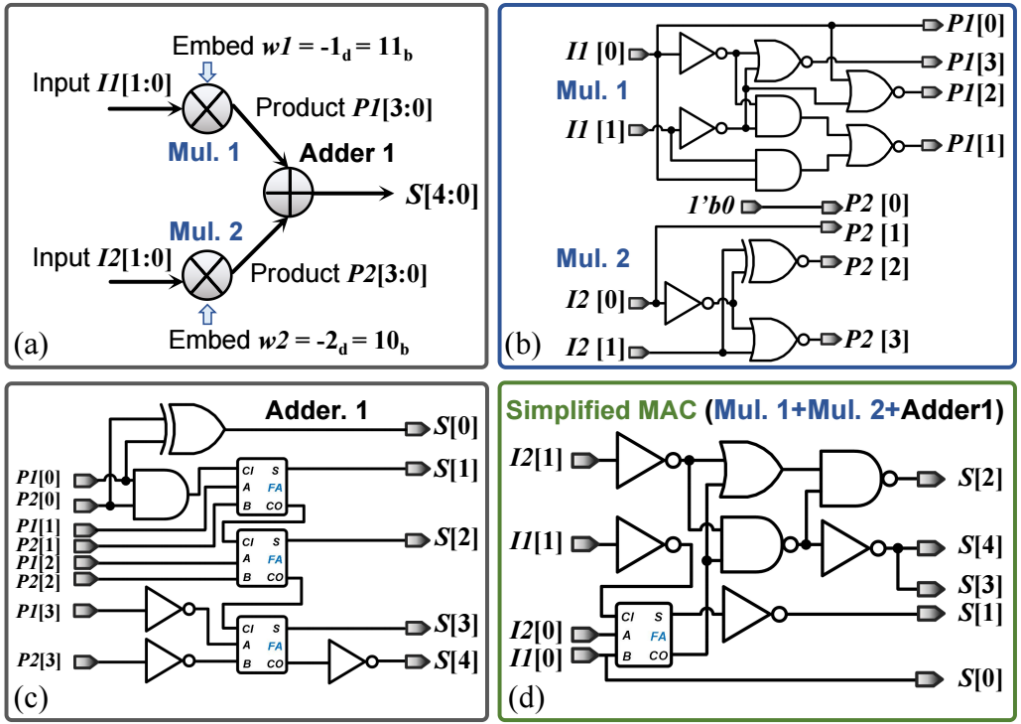




Logic circuit of a 2-bit signed multiplier. (a) The original circuit; (b) The logic circuit simplified with a fixed quantization weight (decimal: -2, binary: 10).

- Use the fixed weights after training to simplify the logic of the multipliers at neurons.
- Example of a 2-bit signed multiplier after embedding a weight:
  - Delay: ↓57.72% Power consumption: ↓66.12% Number of transistors: ↓60%

Logic Implementation: Adder and MAC



(a) MAC operations at a neuron; (b) 2-bit signed multipliers simplified with the fixed quantized weights; (c) 4-bit signed adder circuit before logic simplification, FA is a 1-bit full adder; (d) Circuit of the simplified MAC.

MAC after weight embedding:

- Delay: ↓70.07%
- Power consumption: ↓60.94%
- Transistors: ↓65%

LLM-Driven Circuit Design

- **Overview of LLM in front-end design**
- LLM-assisted testbench generation
- LLM-assisted circuit generation
- LLM-assisted high-level synthesis

VHDL Code Generation Example

Give me a 4-bit adder in VHDL with the highest performance. Change the output of 0+0 to 1.

To design a **4-bit high-performance adder** in VHDL that also **outputs 1 instead of 0 when A = 0, B = 0, and Cin = 0**, we need to:

- Use a **Carry Look-Ahead Adder (CLA)** for high performance.
- Add **custom logic** to force the result to 0001 when the sum is normally 0000.

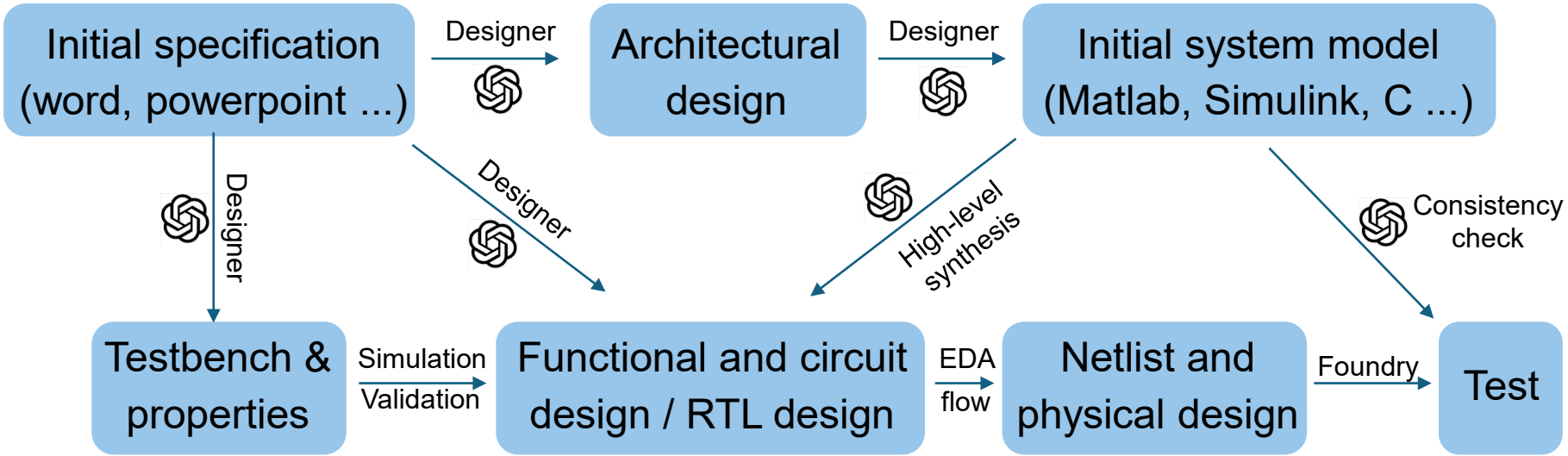
✅ Summary

- This adder is high-speed due to parallel CLA logic.
- It modifies only one condition — when everything is zero, it forces Sum = "0001".
- All other operations behave exactly like a normal 4-bit adder.

4-Bit CLA Adder with Output Override

```
begin
    -- Initial carry-in
    C(0) <= Cin;

    -- Generate and Propagate
    G <= A and B;
    P <= A xor B;

    -- Carry Look-Ahead Logic
    C(1) <= G(0) or (P(0) and C(0));
    C(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and C(0));
    C(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or
            (P(2) and P(1) and P(0) and C(0));
    C(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or
            (P(3) and P(2) and P(1) and G(0)) or
            (P(3) and P(2) and P(1) and P(0) and C(0));

    -- Standard sum output
    rawSum(0) <= P(0) xor C(0);
    rawSum(1) <= P(1) xor C(1);
    rawSum(2) <= P(2) xor C(2);
    rawSum(3) <= P(3) xor C(3);

    -- Custom override: if A = 0000, B = 0000, Cin = 0 => force Sum = 0001
    process(A, B, Cin, rawSum)
    begin
        if (A = "0000" and B = "0000" and Cin = '0') then
            Sum <= "0001";
        else
            Sum <= rawSum;
        end if;
    end process;

    -- Carry out
    Cout <= C(4);
end Behavioral;
```

LLMs for Chip Design

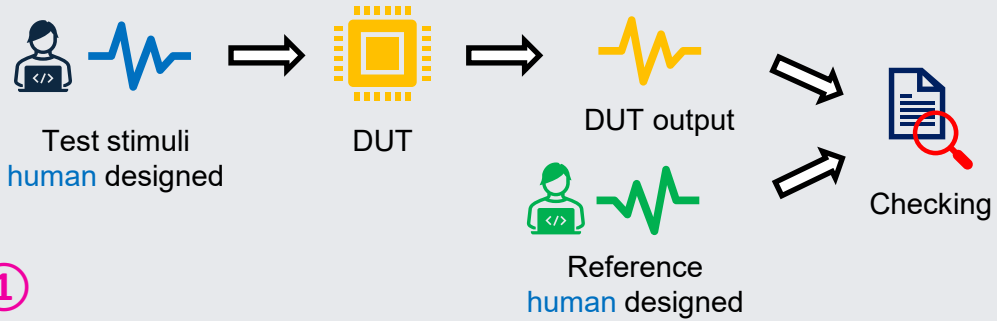


LLM-Driven Circuit Design

- Overview of LLM in front-end design
- **LLM-assisted testbench generation**
- LLM-assisted circuit generation
- LLM-assisted high-level synthesis

TestBench Auto-Generation: Targets and Challenges

Testbench-based hardware simulation

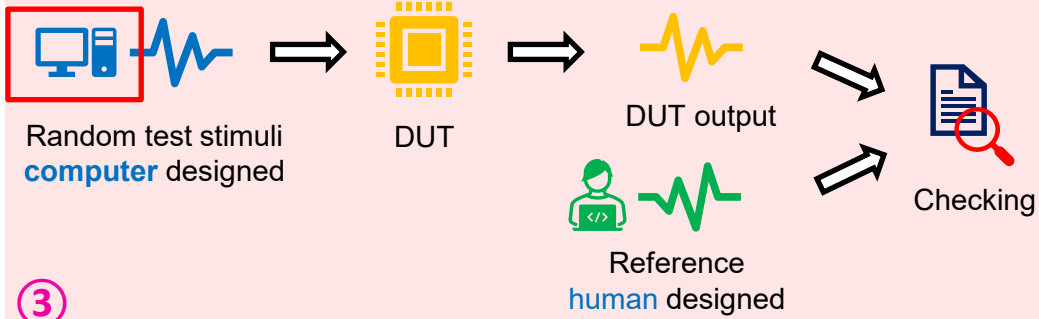


①

Goal of testbench auto-generation: Given the **SPEC** of DUT in natural language, automatically generate the **Testbench** for RTL design validation

②

Previous automatic testbench generation



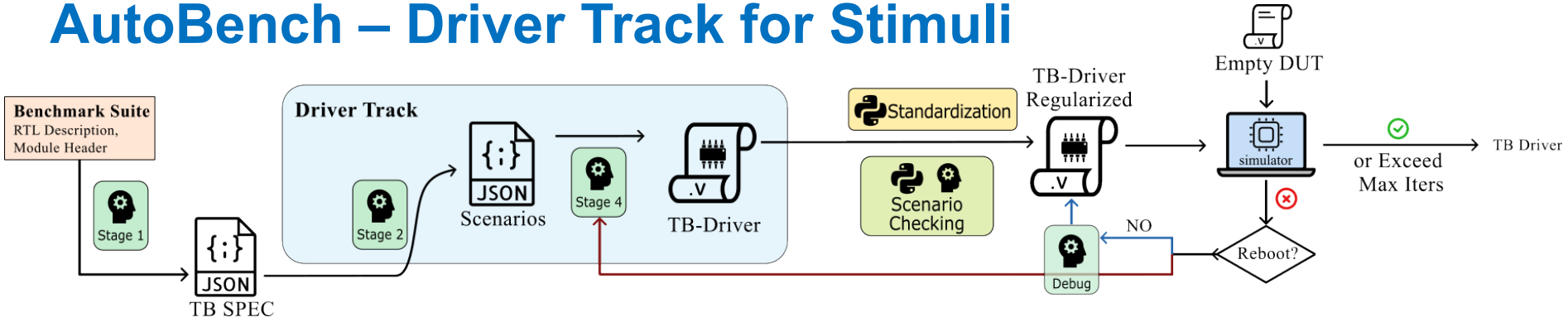
③

Challenges of previous work

1. Reference signals are still **manually** designed
2. Random test stimuli are **hard to understand** for debugging
3. A large number of stimuli for a good coverage

④

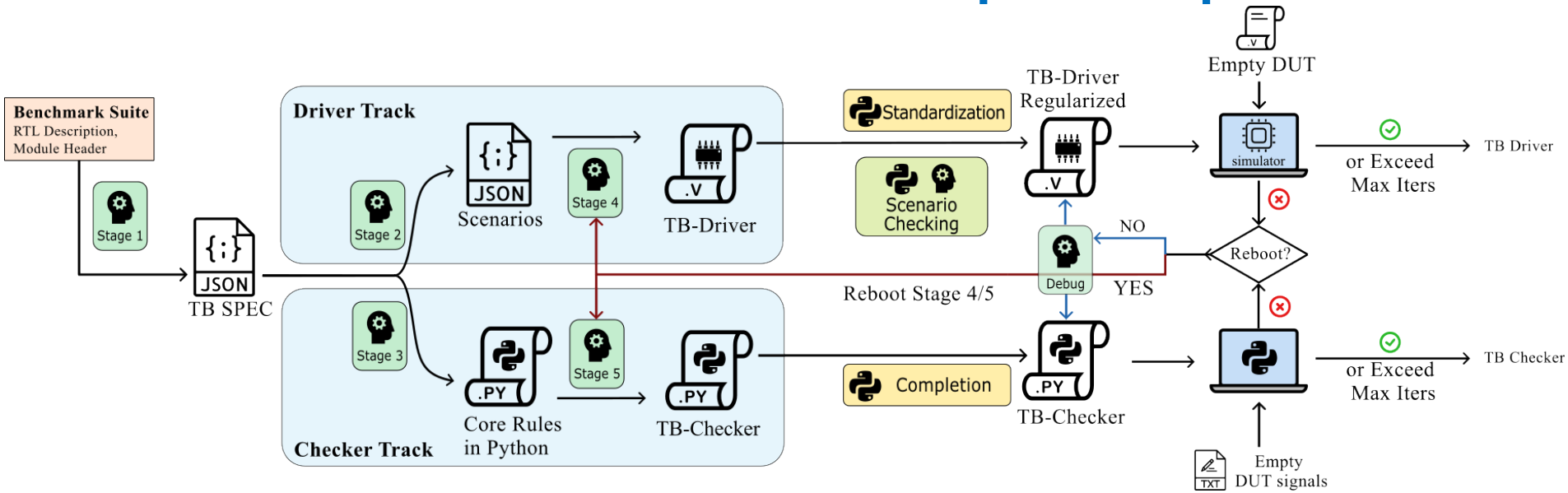
AutoBench – Driver Track for Stimuli



- Stage 1: Convert DUT Spec to TB Spec
- Stage 2: Force LLM to generate test scenarios to avoid laziness
- Stage 4: Generate Verilog driver as input to the DUT
- Scenario checking: Check whether all scenarios are covered and rerun TB generation with partial TB when necessary.

R. Qiu, L. Zhang, R. Drechsler, U. Schlichtmann, B. Li, “AutoBench: Automatic Testbench Generation and Evaluation Using LLMs for HDL Design,” ACM/IEEE International Symposium on Machine Learning for CAD, 2024, Best Artifact Award

AutoBench – Checker Track for Output Comparison



- Python used for checker to reduce similar errors in both driver and checker
- Stage 3: LLM generates the core checking rules in Python
- Stage 5: Checker code generation
- Driver and check together form a complete hybrid testbench.

AutoBench Example

[problem description]/Spec:
Build a combinational circuit with 100 inputs, in[99:0].
There are 3 outputs:
// (1) out_and: output of a 100-input AND gate.
// (2) out_or: output of a 100-input OR gate.
// (3) out_xor: output of a 100-input XOR gate.

Prompt:
1. Your task is to write a verilog testbench for a verilog RTL module code ... (omitted due to space constraints)

2. You are in the first stage. In this stage, please summarize the technical details of the DUT and give me a technical specification of the testbench generation task, so we can use it to design its corresponding testbench.

3. ... (omitted due to space constraints)

4. Your technical specification should include: ... (omitted due to space constraints)

5. Below is the information including the problem description and the DUT header:
[**Benchmark Suite**]

Your response must be in JSON form. example: { "important data": "...", # type: string. If no table, state transition or other direct data, leave this with "" "technical specifications": ["...", "...", ...] # each element of the list is one specification string, the starting of the string is its index . }

LLM's Output:
[testbench specification]:

{"important data": "",
"technical specifications": ["1. DUT Module Description: The Device Under Test (DUT) is defined in the Verilog module named 'top_module'. It contains a single input bus of width 100 bits (in[99:0]) and three separate output signals: out_and, out_or, and out_xor. These outputs represent the computed AND, OR, and XOR operations of all 100 inputs respectively.",
"2. Input Specification: The DUT accepts a 100-bit input vector. Each bit of this vector can independently be either 0 or 1. Comprehensive testing will involve applying various combinations of zeros and ones to these inputs."
"3. ...", ...]} (omitted due to space constraints)

Stage 1 TB Spec

Example: Stimuli

[problem description]:
Build a combinational circuit with 100 inputs, in[99:0]. There are 3 outputs:
// (1) out_and: output of a 100-input AND gate.
// (2) out_or: output of a 100-input OR gate.
// (3) out_xor: output of a 100-input XOR gate.

```
top_module DUT (.in(in),.out_and(out_and),.out_or(out_or)
     ,.out_xor(out_xor));
initial begin
    file_descriptor = $fopen("TBout.txt", "w");
    // Scenario 1: All bits are zero
    in = 100'b0;
    #10 $fdisplay(file_descriptor, "scenario: 1, in = %d,
          out_and = %d, out_or = %d, out_xor = %d", in,
          out_and, out_or, out_xor);
    // Scenario 2: All bits are one
    in = ~100'b0;
    #10 $fdisplay(file_descriptor, "scenario: 2, in = %d,
          out_and = %d, out_or = %d, out_xor = %d", in,
          out_and, out_or, out_xor);
    // Scenario 3: Alternating bits starting with 0
    in = 100'b0101010101_0101010101_0101010101_
    0101010101_0101010101_0101010101_0101010101_
    0101010101_0101010101_0101010101;
    #10 $fdisplay(file_descriptor, "scenario: 3, in = %d,
          out_and = %d, out_or = %d, out_xor = %d", in,
          out_and, out_or, out_xor);
... // omitted
```

Example:Checker

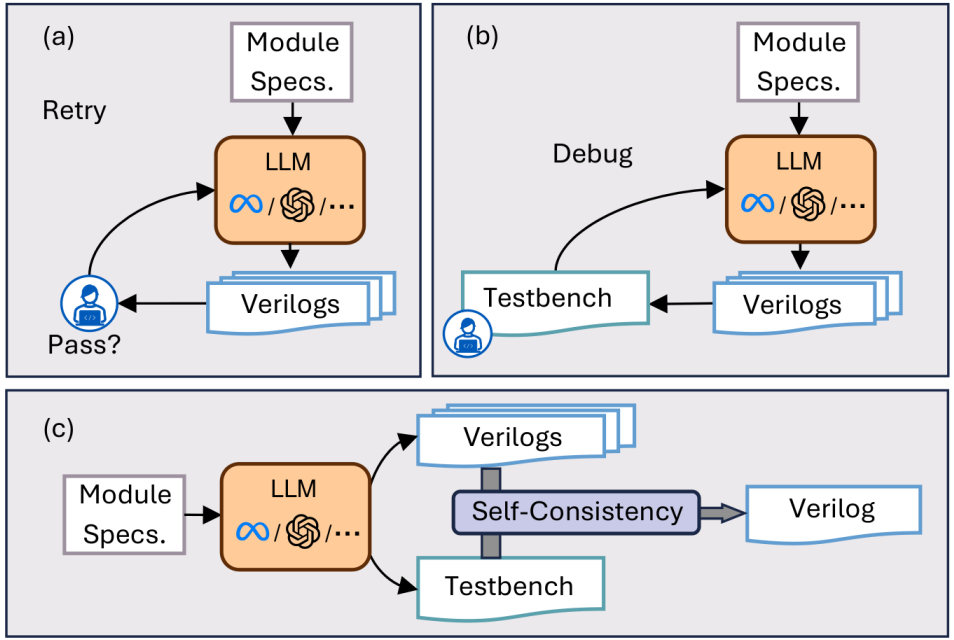
[problem description]:
Build a combinational circuit with 100 inputs, in[99:0]. There are 3 outputs:
// (1) out_and: output of a 100-input AND gate.
// (2) out_or: output of a 100-input OR gate.
// (3) out_xor: output of a 100-input XOR gate.

```
import numpy as np
def calculate_out_and(input_vector):
    return np.all(input_vector == 1)
def calculate_out_or(input_vector):
    return np.any(input_vector == 1)
def calculate_out_xor(input_vector):
    return np.sum(input_vector) % 2 == 1
def check_dut(test_vectors):
    failed_scenarios = [[
    for idx, scenario in enumerate(test_vectors):
        input_value = scenario['in'[
        input_vector = np.array([int(x) for x in np.
              binary_repr(input_value, width=100)[)
        expected_and = calculate_out_and(input_vector)
        expected_or = calculate_out_or(input_vector)
        expected_xor = calculate_out_xor(input_vector)
        actual_and = bool(scenario['out_and'[)
        actual_or = bool(scenario['out_or'[)
        actual_xor = bool(scenario['out_xor'[)
        if (expected_and != actual_and or expected_or
              != actual_or or expected_xor !=
              actual_xor):
            failed_scenarios.append(idx)
    return failed_scenarios
```

LLM-Driven Circuit Design

- Overview of LLM in front-end design
- LLM-assisted testbench generation
- **LLM-assisted circuit generation**
- LLM-assisted high-level synthesis

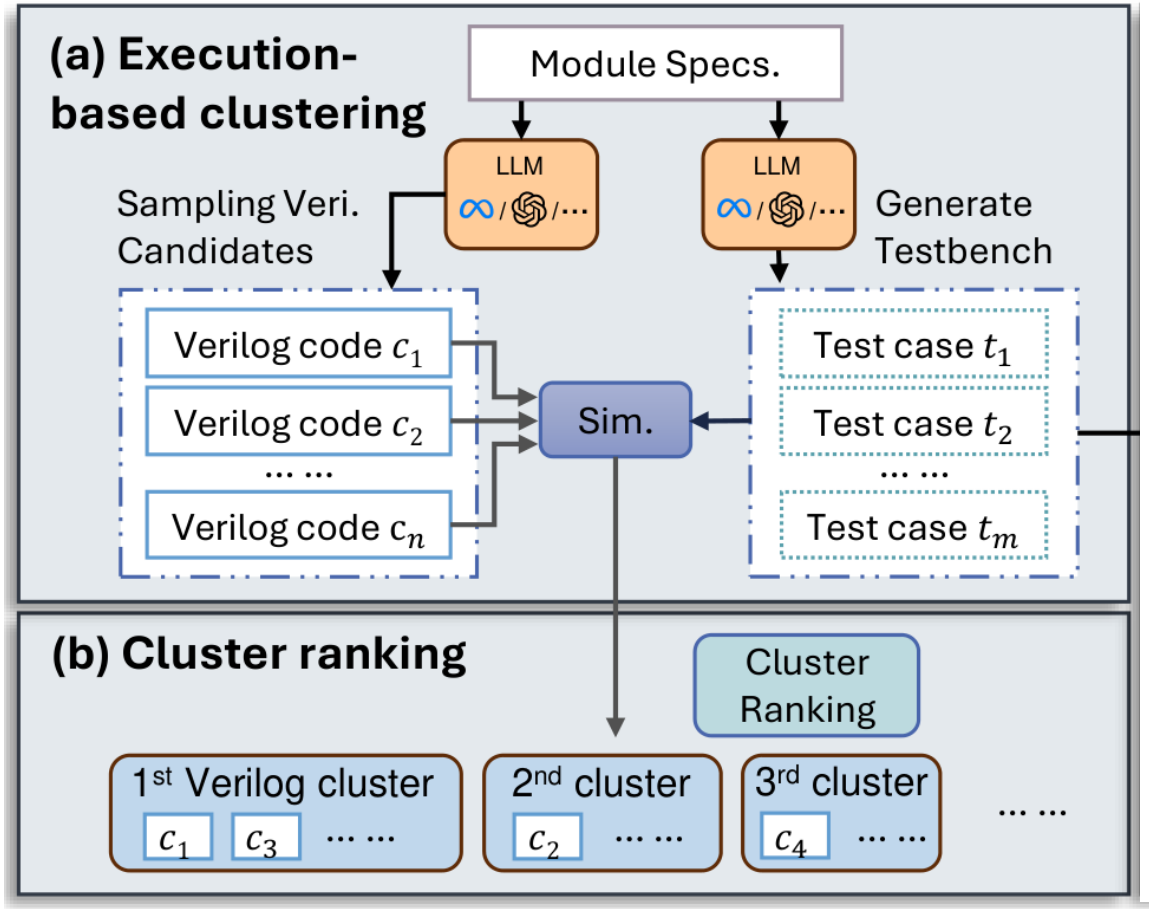
LLMs for Circuit Generation



Comparison between (a) direct sampling, (b) debugging, and (c) VRank (Proposed).

- Quality of generated code is unstable due to the probabilistic nature of LLMs.
- State-of-the-art work:
  - Finetune
  - Repeatedly generate and let human check for bugs (a)
  - Use golden testbench to provide feedback info (b)
- Vrank/VFocus: Self-consistency enhancement uses automatically generated testbenches to select code cases.

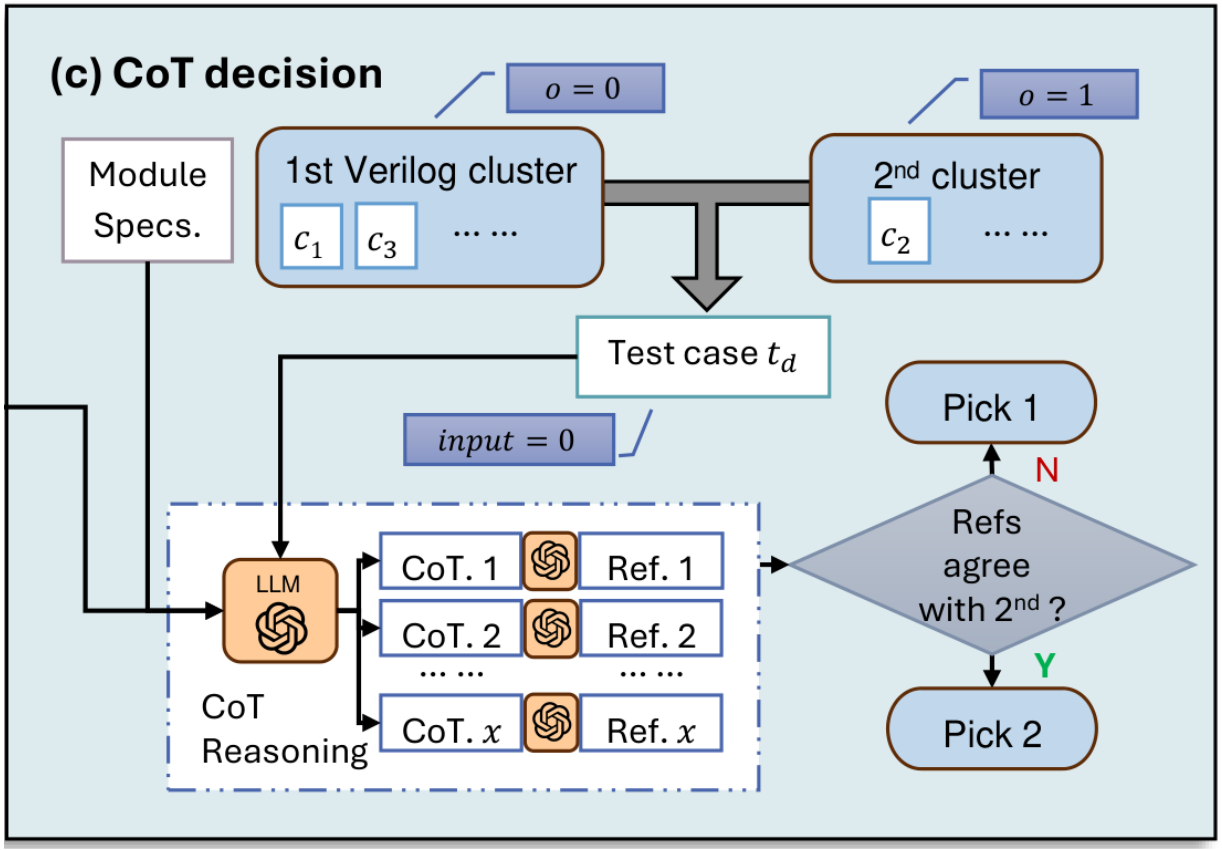
VRank Framework



(a) Multiple code cases generated and simulated by automatically generated testbench.

(b) Code cases generating the same output w.r.t testbench are grouped into one cluster.

Test Refinement



- Top-ranked cluster may not produce the best code due to errors in both codes and testbenches.
- The testbench that differentiates top two clusters and the module's specification are fed into LLM to refine the testbench output.
- Use the testbench results to rank clusters again and select the best code.

Experimental Results

Model	Dataset	Baseline			Framework Pass@1(Pass@1 increase)		
		Pass@1	Pass@2	Pass@3	VRank	Pre+VRank	VFocus
Deepseek-R1	Human	66.0%	70.9%	72.9%	79.2% (+13.2%)	84.7%(+18.7%)	87.0%(+21.0%)
o3-mini		65.3%	70.4%	72.4%	77.4%(+12.1%)	84.2%(+18.9%)	85.6%(+20.3%)
QwQ-32B		51.7%	58.1%	61.1%	69.1%(+17.4%)	74.4%(+22.7%)	77.1%(+25.4%)
Deepseek-R1	CMB(81)	83.1%	87.7%	89.4%	94.9%(+11.8%)	94.8%(+11.7%)	95.0%(+11.9%)
o3-mini		78.6%	83.7%	85.3%	88.9%(+10.3%)	94.1%(+15.5%)	94.3%(+15.7%)
QwQ-32B		70.5%	77.3%	80.0%	88.2%(+17.7%)	93.6%(+23.1%)	93.3%(+22.8%)
Deepseek-R1	SEQ(75)	47.6%	52.3%	55.2%	62.2%(+14.6%)	72.4%(+24.8%)	78.5%(+30.9%)
o3-mini		50.9%	56.0%	58.4%	63.8%(+12.9%)	73.4%(+22.5%)	76.2%(+25.3%)
QwQ-32B		31.5%	37.4%	40.6%	48.6%(+17.1%)	53.7%(+22.2%)	59.6%(+28.1%)

- Dataset: VerilogEval-Human, 156 Verilog problems
- Models: Deepseek-r1,QwQ-32b, and o3-mini

Zhuorui Zhao et al., VRank: Enhancing Verilog Code Generation from Large Language Models via Self-Consistency, ISQED 2025, Best Paper Award
Zhuorui Zhao et al, “VFocus: Better Verilog Generation from Large Language Model via Focused Reasoning,” SOCC 2025

LLM-Driven Circuit Design

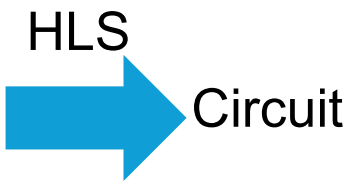
- Overview of LLM in front-end design
- LLM-assisted testbench generation
- LLM-assisted circuit generation
- **LLM-assisted high-level synthesis**

High-Level Synthesis

C/C++ programming for circuit design

```
void DFS(TreeNode *current) {
    unsigned int m = visit(current->value);
    DFS(current->left);
    DFS(current->right);}
```

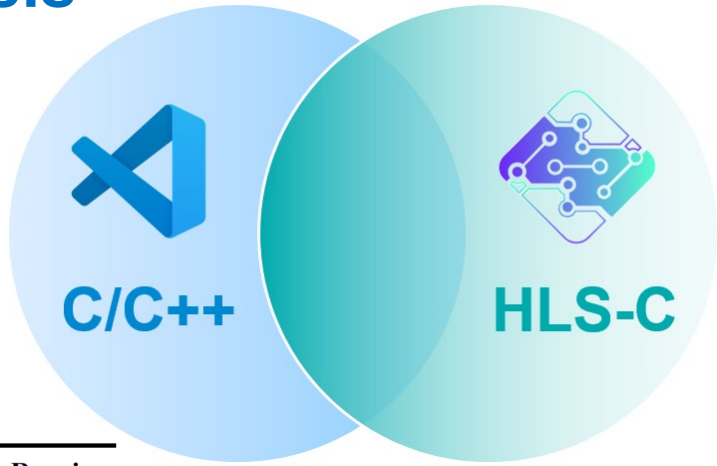
Recursion



Challenges in High-Level Synthesis

HLS-C is not standard C/C++:

- HLS tools only support a subset of C/C++
- Writing HLS-C or converting regular C/C++ requires knowledge in both hardware and software



Incompatibility Type	Typical Symptom (# Error / Warning:)	Corresponding Repair
Pointer	Unsupported feature 'pointers-to-pointers' (CIN-243)	Convert to array access
Dynamic Array	Unsupported feature 'variable length array' (CIN-15)	Specify the array size
Recursion	Unsupported feature 'recursion' (CIN-15)	Replace with a loop
Bit Width	Using ac_int<> need consider bit accurate (CIN-46)	Optimize the bit width
Boolean Operation	Incrementing a bool value is deprecated (CRD-708)	Convert to integer type
Incomplete Statement	Expected a statement (CRD-127)	Complete the statement
Unsupported Struct	Unsupported structs create deadlocks. (HIER-10)	Add explicit constructors
Exception-Handling	Out-of-bounds read/write access (CRD-175)	Add a bound check

HLS Repair Example

```
void DFS(TreeNode *current) {
    unsigned int m = visit(current->value);
    DFS(current->left);
    DFS(current->right);}
```

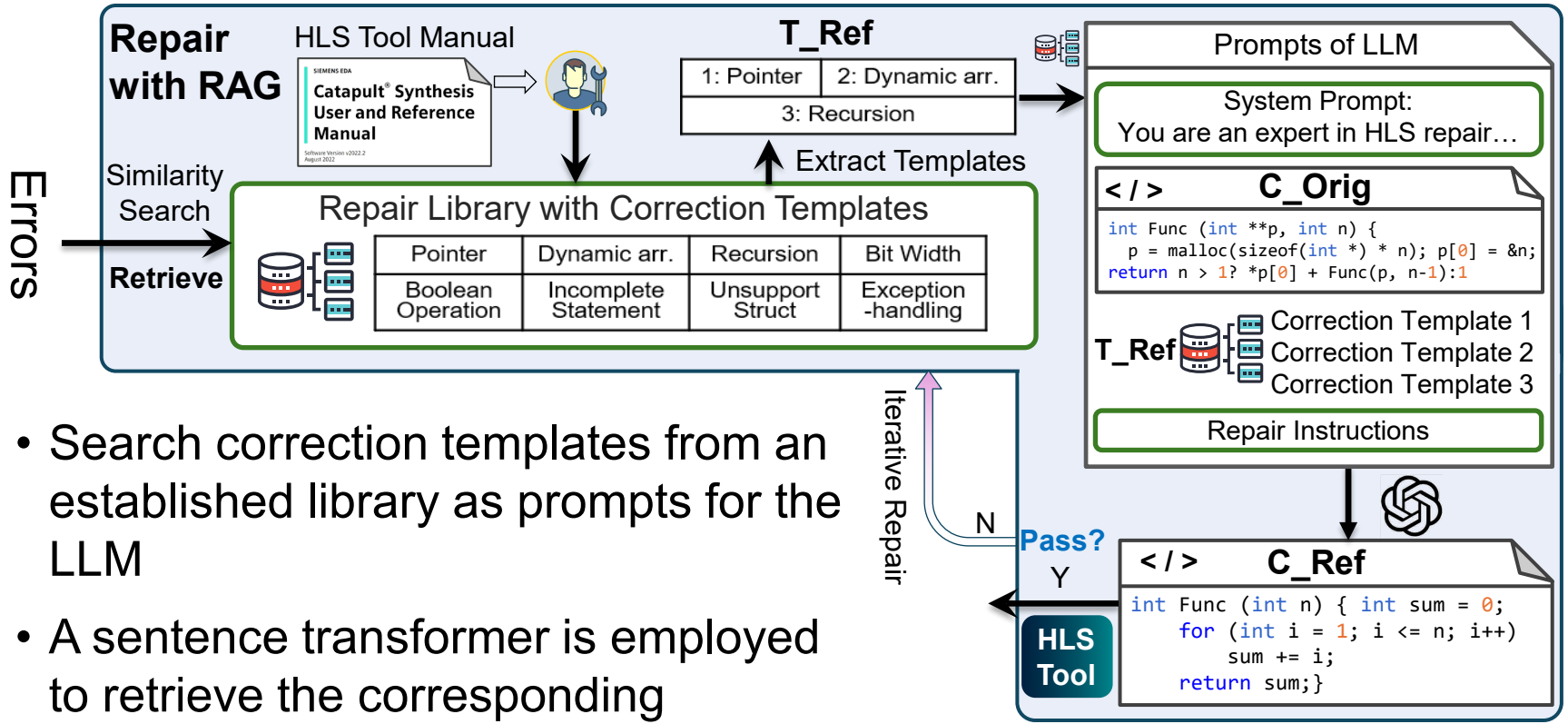
Recursion

Manual conversion of C code into HLS-C can be tedious and error-prone. (This example is converted by LLM automatically.)

Loop Iteration

```
void DFS(unsigned int rootIndex) {
    Record stack[MAX]; int Tstack = 0;
    stack[Tstack++] = {rootIndex, false, false};
  while (Tstack > 0) {
     …
    Record &top = stack[--Tstack];
    if (!top.L_Visited && Pool[top.Index].left != MAX) {
        ac_int<9, false > m = visit(Pool[top.Index].value);
        top.L_Visited = true;
        stack[Tstack++] = {Pool[top.Index].left, false, false};
      } else if (!top.R_Visited…)}}
```

Automatic Repair Flow with RAG



- Search correction templates from an established library as prompts for the LLM
- A sentence transformer is employed to retrieve the corresponding correction template.

Pass Rate of HLS

Comparison with script-based solutions and GPT-4 Turbo baseline

Type	Benchmark	Tradi-tional [6] [7]	Baseline (Pass Rate %) Compi.	Baseline (Pass Rate %) Simu.	Proposed (Pass Rate %) Compi.	Proposed (Pass Rate %) Simu.	Type	Benchmark	Tradi-tional [6] [7]	Baseline (Pass Rate %) Compi.	Baseline (Pass Rate %) Simu.	Proposed (Pass Rate %) Compi.	Proposed (Pass Rate %) Simu.
T1: Pointer	1: Double Pointer	×/×	60	53.33	73.33	73.33	T5:	13: Support Vector Mac.	×/×	46.67	46.67	93.33	93.33
	2: Hash Table	✓/✓	93.33	93.33	100	100	Boolean	14: Fourier Transform	×/×	40	33.33	80	80
	3: Deep Neural Network	×/×	60	60	86.67	80	Operation	15: Color Correction	×/×	46.67	46.67	66.67	66.67
T2: Dynamic Array	4: Linear Programming	✓/×	93.33	86.67	100	100	T6:	16: Fibonacci Sequence	✓/✓	86.67	80	100	100
	5: Binary Tree	×/×	66.67	60	80	80	Incomplete	17: Cyclic Rotation	×/×	40	40	86.67	73.33
	6: K-Nearest Neighbor	×/×	60	46.67	73.33	66.67	Statement	18: AES	×/×	33.33	26.67	60	60
T3: Recursion	7: Linked List	✓/✓	73.33	73.33	93.33	93.33	T7:	19: Data Stream	×/×	66.67	53	73.33	66.67
	8: Depth-First Search	×/×	46.67	40	80	66.67	Unsupport.	20: Longest Increa. Path	×/×	60	60	86.67	86.67
	9: Breadth-First Search	×/×	60	60	73.33	73.33	Struct	21: Max Points on Line	×/×	46.67	40	80	73.33
T4: Bit Width	10: Edge Detection	×/×	53.33	33.33	73.33	60	T8:	22: QR Decomposition	×/×	66.67	60	80	80
	11: Greedy Algorithm	×/×	46.67	46.67	86.67	80	Exception-	23: Dump Filter	×/×	33.33	33.33	66.67	66.67
	12: Bubble Sort	✓/✓	100	100	100	100	Handling	24: Turbo Encoder	×/×	73.33	66.67	93.33	93.33

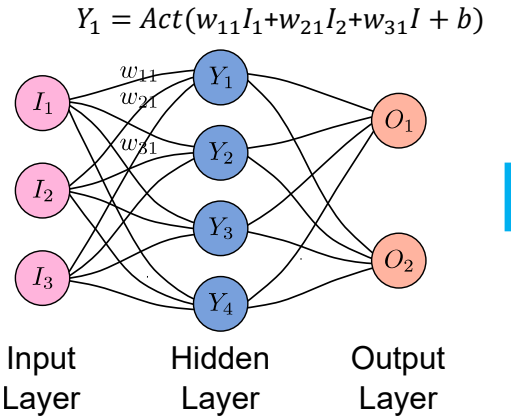
~20% pass rate improvement

K. Xu et al., “Automated C/C++ Program Repair for High-Level Synthesis via Large Language Models,” MLCAD 2024

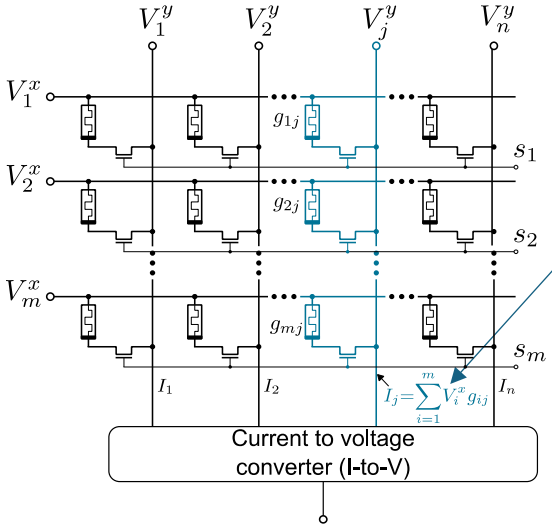
In-Memory Computing

- **Variation compensation**
- Basis computing without reprogramming
- Optical circuit test

Analog Hardware Acceleration of Neural Networks



Neural network with three layers

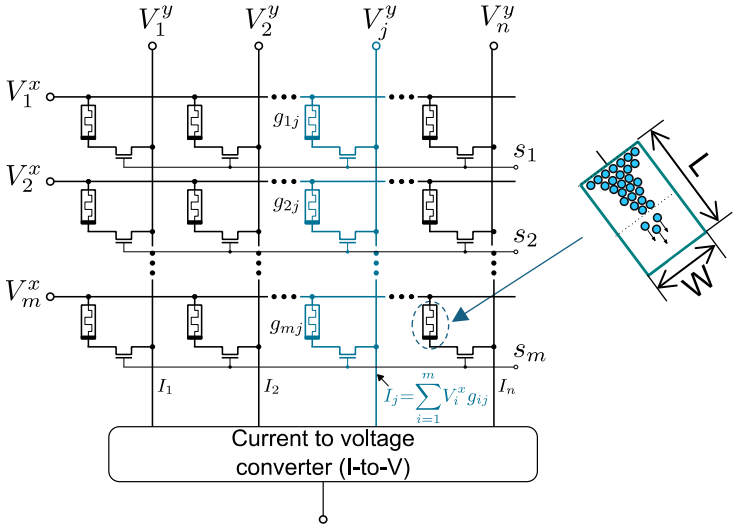


- Multiplication implemented as the product of voltage and conductance
- Accumulated current as the result of sum

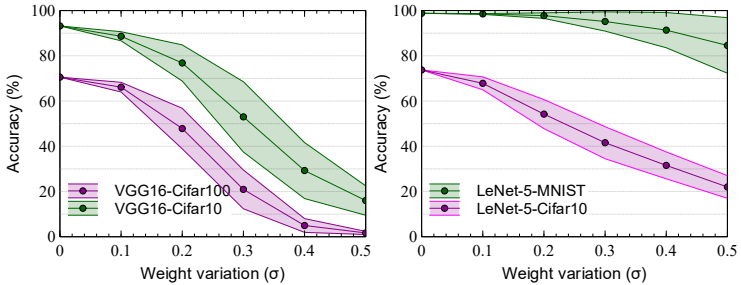
Accelerate neural networks with RRAM (Resistive Random Access Memory) crossbar

- Challenges in analog acceleration: Variations and programming

Variations in RRAM Crossbars

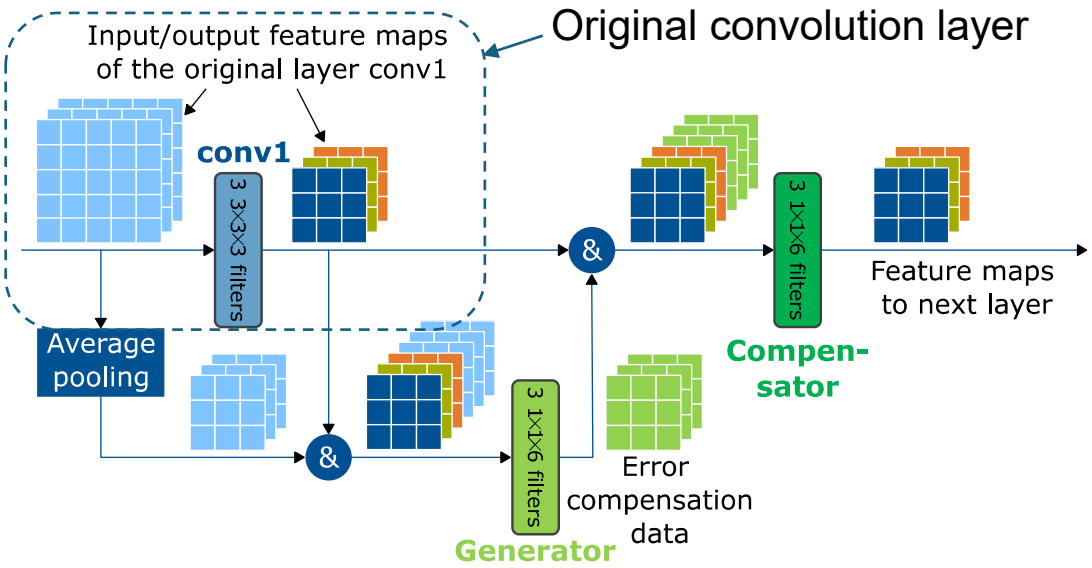


- Weight programmed into RRAM cells inaccurately due to variations and noise
- Such uncertainties lead to large accuracy degradation.



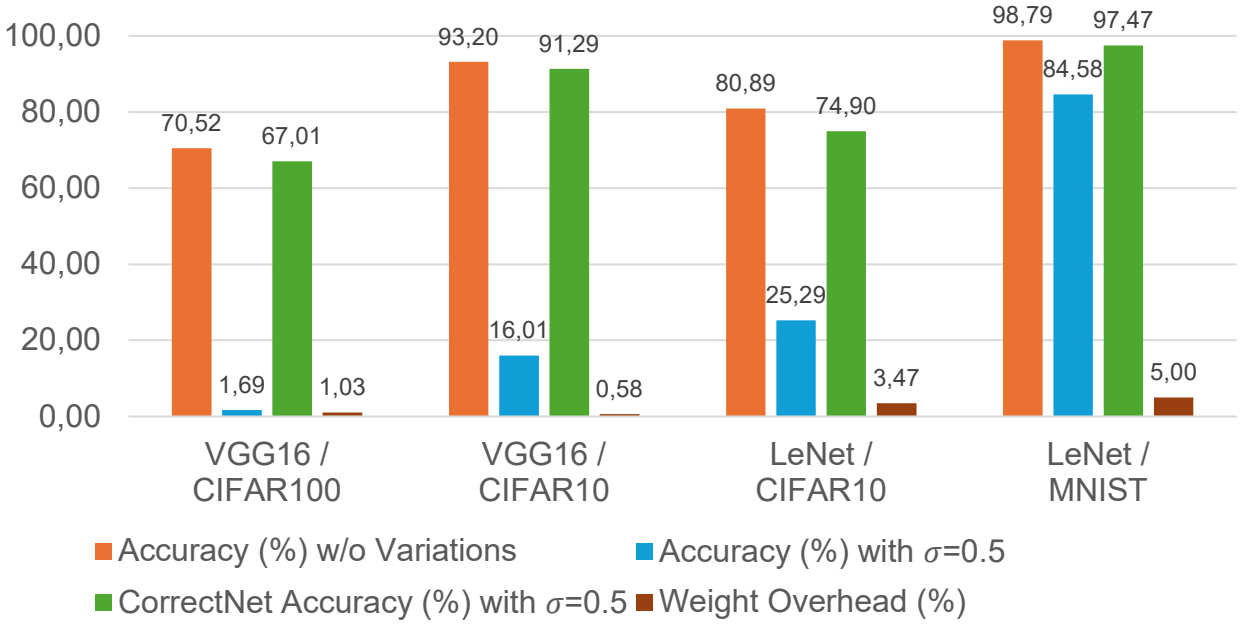
D. Niu, Y. Chen, C. Xu, and Y. Xie, "Impact of process variations on emerging memristor," *ACM/IEEE Des. Autom. Conf. (DAC)*, 2010.
B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, "Vortex: Variation-aware training for memristor X-bar," *ACM/IEEE Des. Autom. Conf. (DAC)*, 2015.

Structural Error Compensation Countering Uncertainties



- Generator: generate error compensation data
- Compensator: correct the erroneous feature map with the compensation data
- Two-step training for error compensation

Accuracy Comparison under Uncertainties

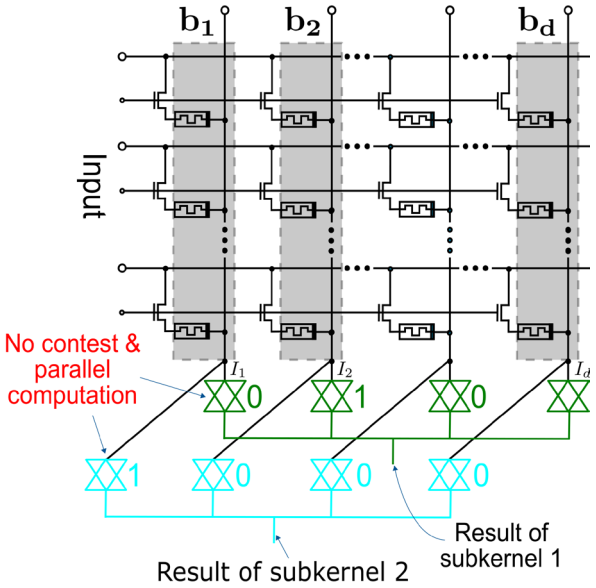


A. Eldebiky, L. Zhang, G. Böcherer, B. Li, and U. Schlichtmann, "CorrectNet: Robustness Enhancement of Analog In-Memory Computing for Neural Networks by Error Suppression and Compensation," *Des. Autom. and Test in Europe (DATE),* 2023, Best Paper Award Candidate

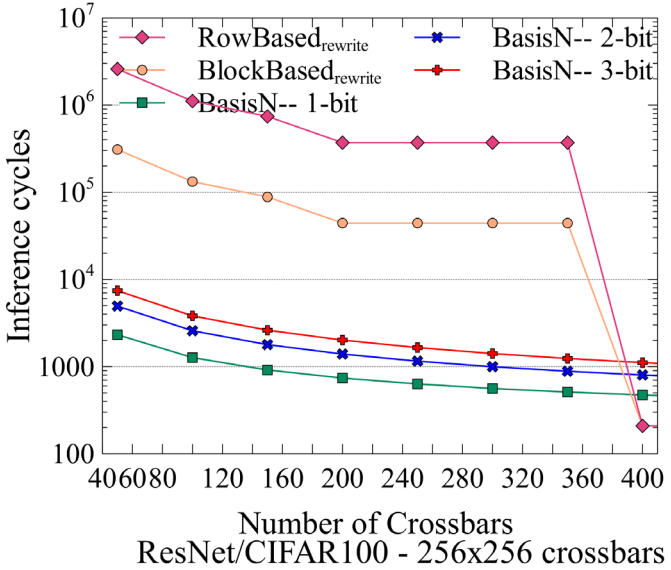
In-Memory Computing

- Variation compensation
- **Basis computing without reprogramming**
- Optical circuit test

Efficient IMC Computing without Reprogramming



IMC computing with pretrained and preprogrammed vectors



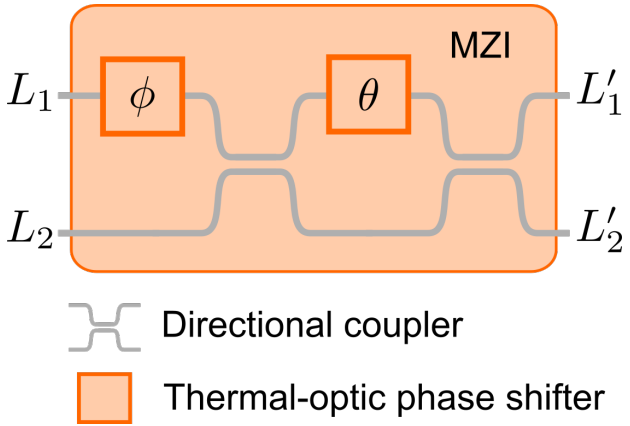
Significant inference cycle reduction with similar accuracy

A. Eldebiky, L. Zhang, X. Yin, C. Zhuo, I. Lin, U. Schlichtmann, B. Li, “BasisN: Reprogramming-Free RRAM-Based In-Memory-Computing by Basis Combination for Deep Neural Networks,” IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2024

In-Memory Computing

- Variation compensation
- Basis computing without reprogramming
- **Optical circuit test**

Computing with Optical Components



- Light signal transformation by Mach-Zehnder Interferometer (MZI)
- Direct coupler: split signal by 50:50; append π/2 in phases of diagonal transmission
- Phase shifter: programmable phase shifters

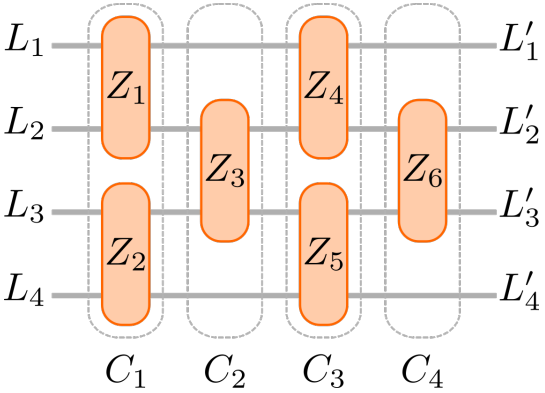
Matrices of phase shifters

Constrained matrix-vector multiplication

$$\begin{bmatrix} L_1'^c \\ L_2'^c \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} e^{i\theta} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix} = ie^{i\theta/2} \begin{bmatrix} e^{i\phi}\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \\ e^{i\phi}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \end{bmatrix} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix} = \mathbf{T} \begin{bmatrix} L_1^c \\ L_2^c \end{bmatrix}$$

Matrices of direct couplers

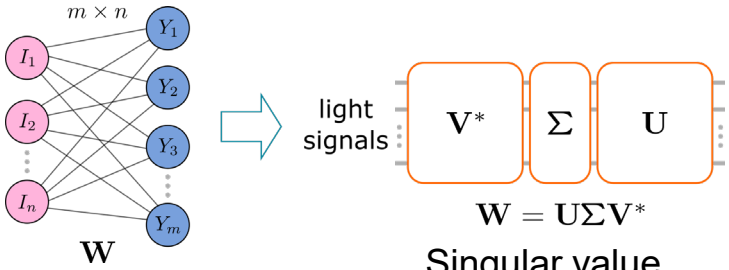
MZIs Networks to Implement Neural Networks



$$\mathbf{T} = \mathbf{T}_{C_4}\mathbf{T}_{C_3}\mathbf{T}_{C_2}\mathbf{T}_{C_1}$$

Multiplication of column matrices formed from the matrices of MZIs

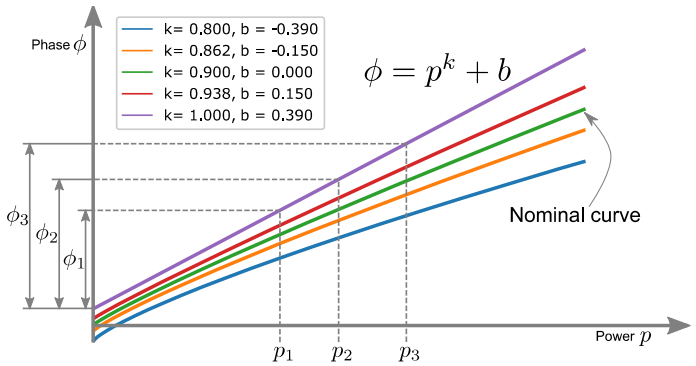
- MZIs connected to implement the multiplication with a larger matrix **T**
- Neural networks mapped onto MZI networks by matrix decomposition



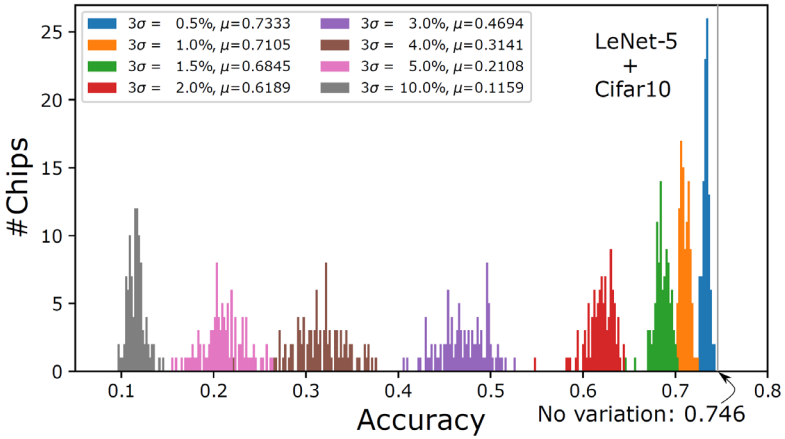
Singular value decomposition (SVD)

Y. Shen, N.C. Harris, and S. Skirloetal, “Deep learning with coherent nanophotonic circuits,” *Nature Photonics*, vol. 11, 2017.

Variations of MZIs



Phase changes *versus* applied power of five MZIs under process variations



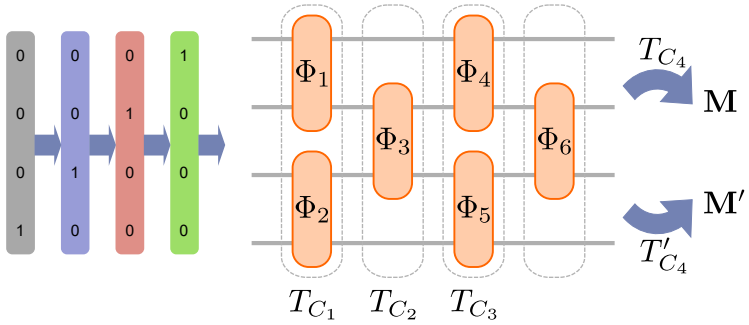
Accuracy degradation of optical neural network under variations (LeNet-5 + Cifar10)

N.C. Harris et al., "Efficient, compact and low loss thermo-optic phase shifter in silicon," *Opt. Express*, 22(9), 2014

Y. Zhu, L. Zhang, B. Li, X. Yin, C. Zhuo, H. Gu, T.-Y. Ho, and U. Schlichtmann, "Countering Variations and Thermal Effects for Accurate Optical Neural Networks," *IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD),* 2020

Recursive Test for MZI Networks

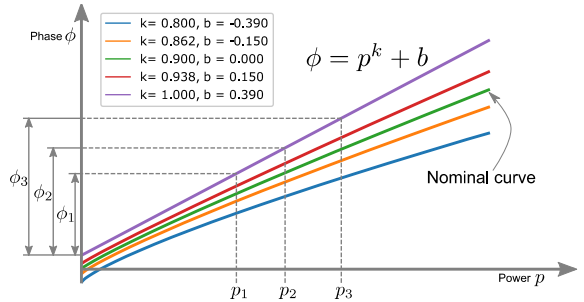
- Test with identity matrix I and changing phases of MZI columns
- MZI variations determined by curve matching and column-wise iterative test



$$M'M^{-1} = \left(T'_{C_4}T_{C_3}T_{C_2}T_{C_1}\right)\left(T_{C_4}T_{C_3}T_{C_2}T_{C_1}\right)^{-1} = T'_{C_4}T_{C_4}^{-1}$$

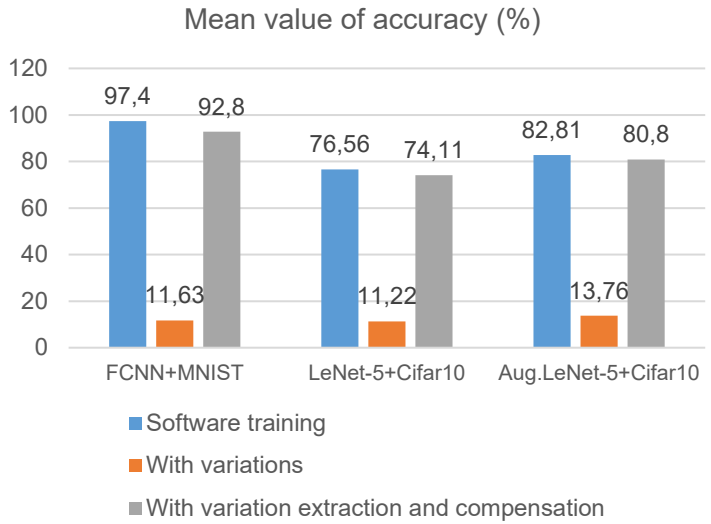
$$= \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & T'_6 & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix}\begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & T_6^* & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} & 0 \\ \mathbf{0} & T'_6T_6^* & \mathbf{0} \\ 0 & \mathbf{0} & 1 \end{bmatrix}$$

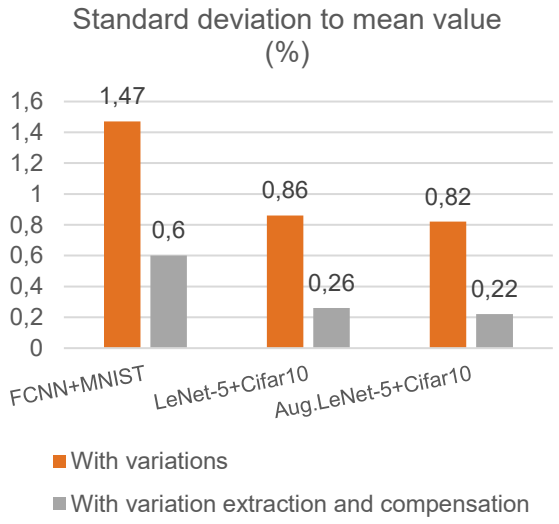
Phase changes in MZI



Curve matching to determine variations

Accuracy Enhancement





#Sampled ONNs: 100; 3σ of the phases at 2π: 20%; Aug. LeNet-5: LeNet-5 with two more convolutional layers

Y. Zhu, L. Zhang, B. Li, X. Yin, C. Zhuo, H. Gu, T.-Y. Ho, and U. Schlichtmann, "Countering Variations and Thermal Effects for Accurate Optical Neural Networks," *IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD),* 2020

Thanks and Questions