Tianen Chen, Taylor Kemp, and Younghyun Kim

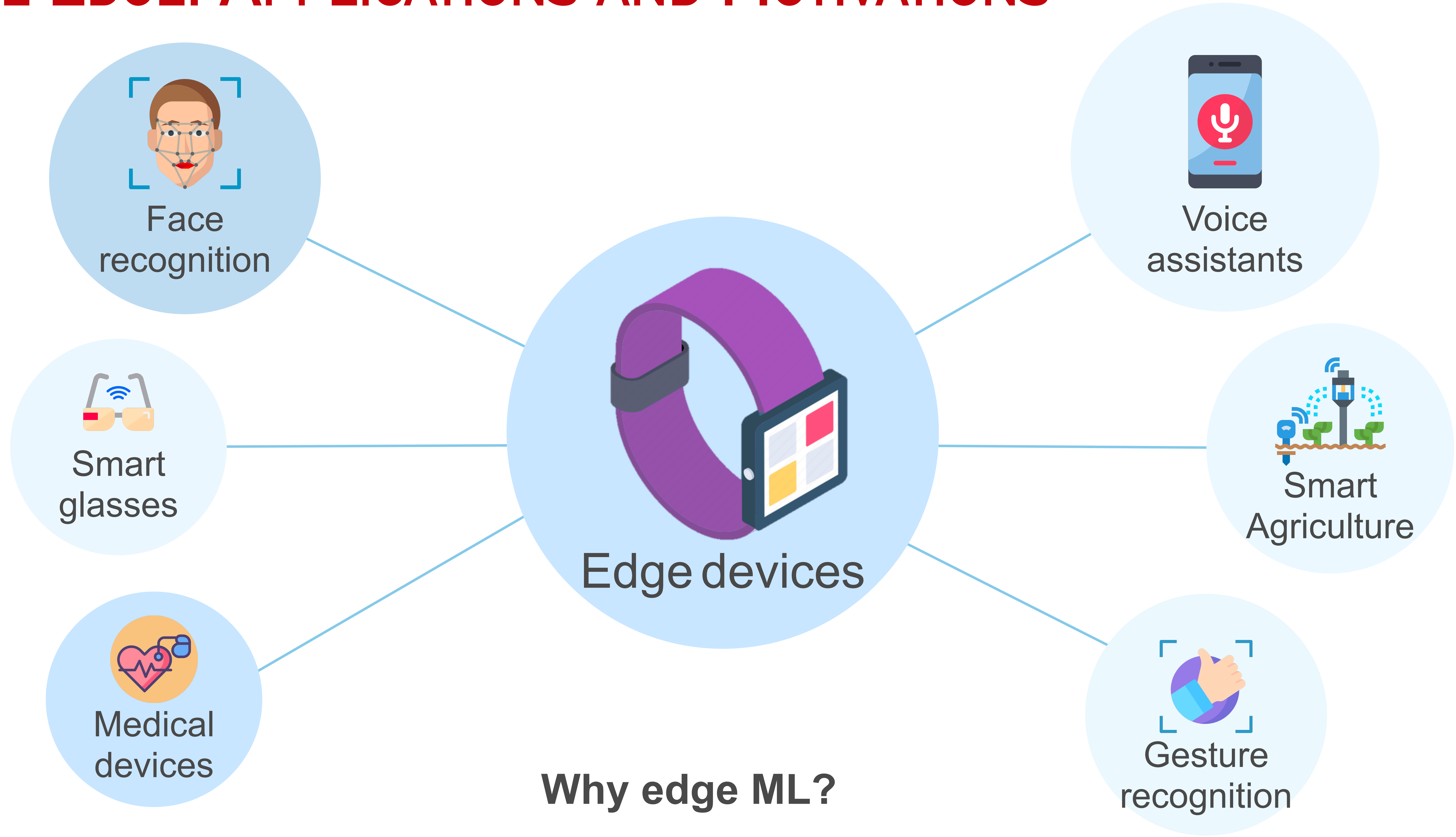
University of Wisconsin–Madison

SynthNet: A High-throughput yet Energy-efficient Combinational Logic Neural Network

ASP-DAC 2022

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

ML AT THE EDGE: APPLICATIONS AND MOTIVATIONS

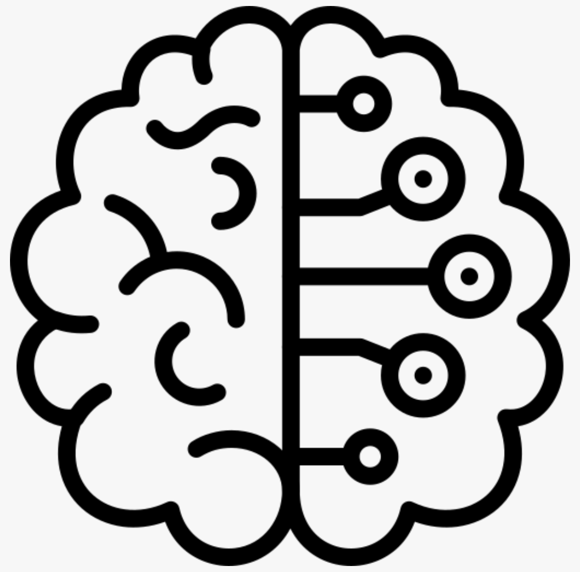


Why edge ML?

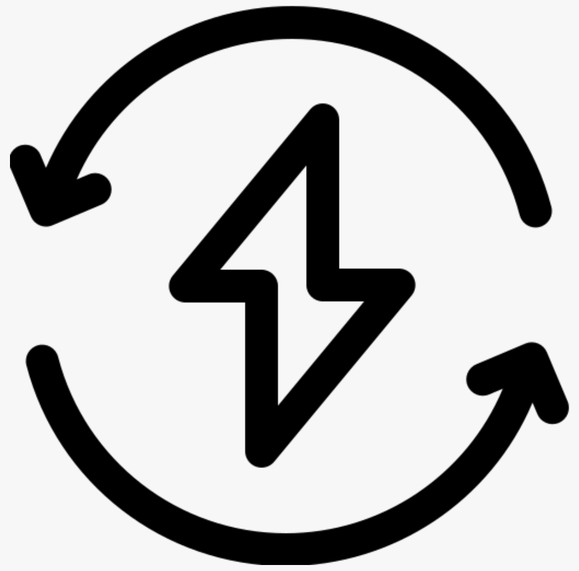
- Less network communication load
- Reduced latency
- Enhanced privacy

ML at the Edge: Challenges

Energy constraints on edge devices



Energy-efficient ML systems



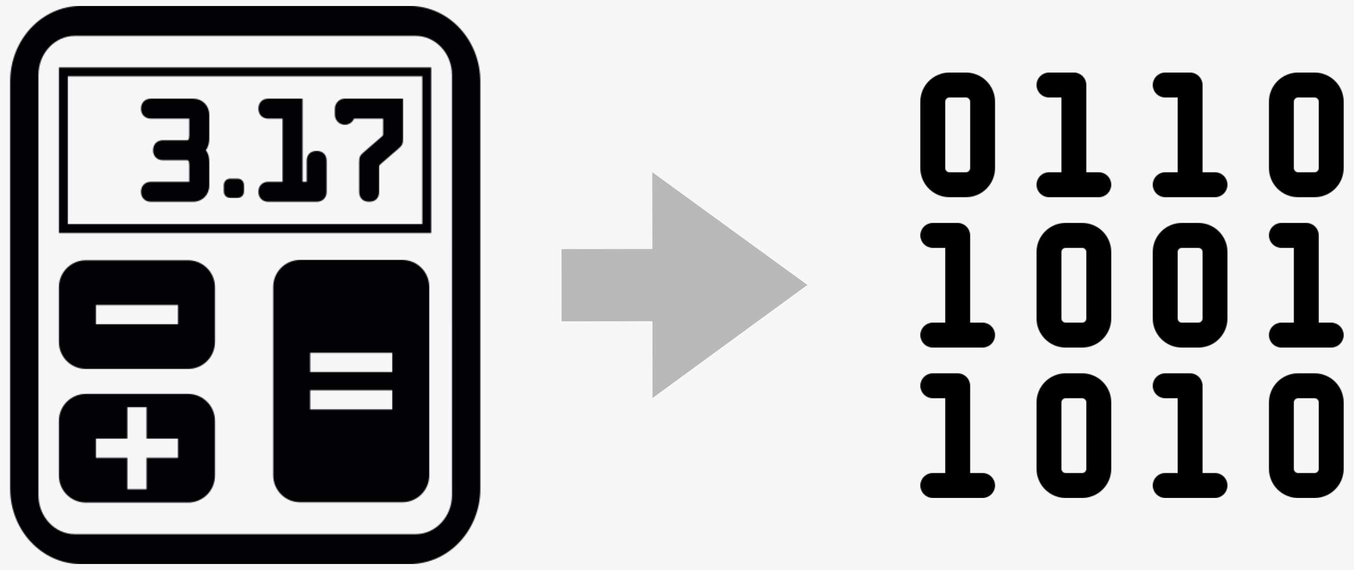
Energy-aware network design

✗ Heavy memory access overheads

- Major source of energy consumption in conventional load-and-compute **neural processing element (NPE)** architectures
- **Multiply-and-accumulate (MAC)** operations dominate memory accesses

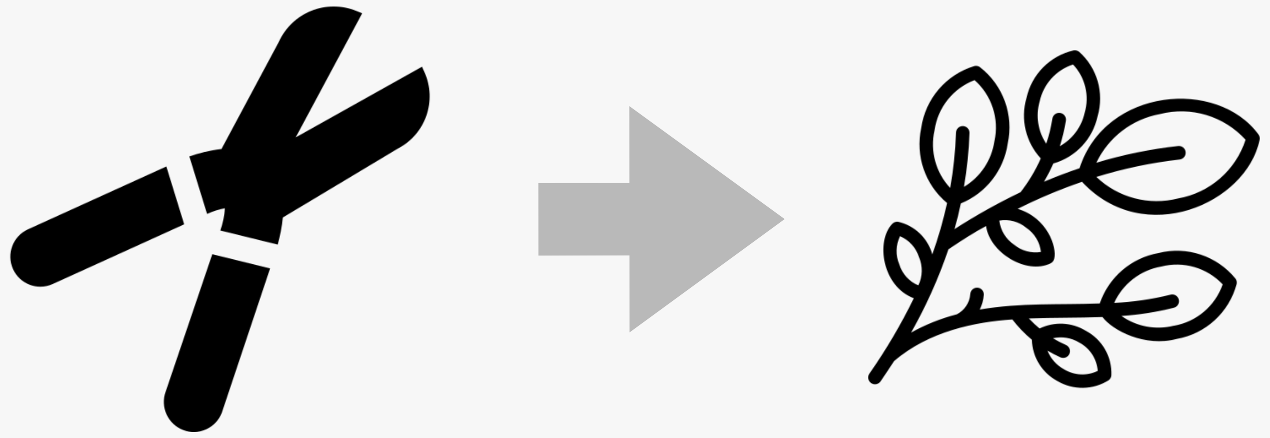
Techniques for Reducing Memory Accesses

Quantization



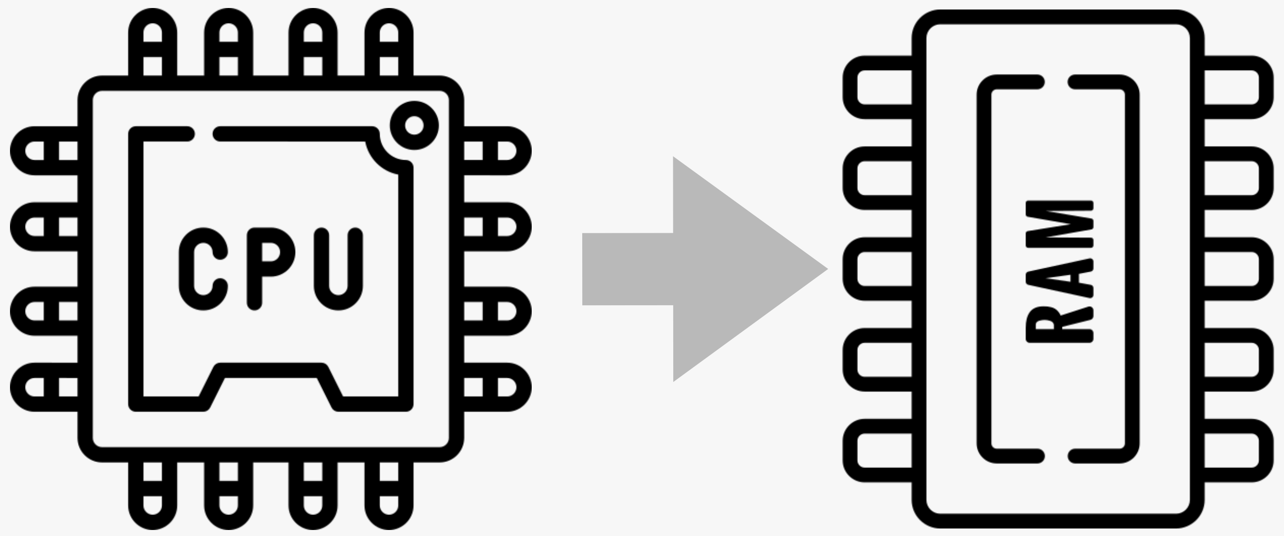
Replace floating-points with numbers with simpler representations

Pruning



Remove connections in order to discriminate against low-magnitude weights

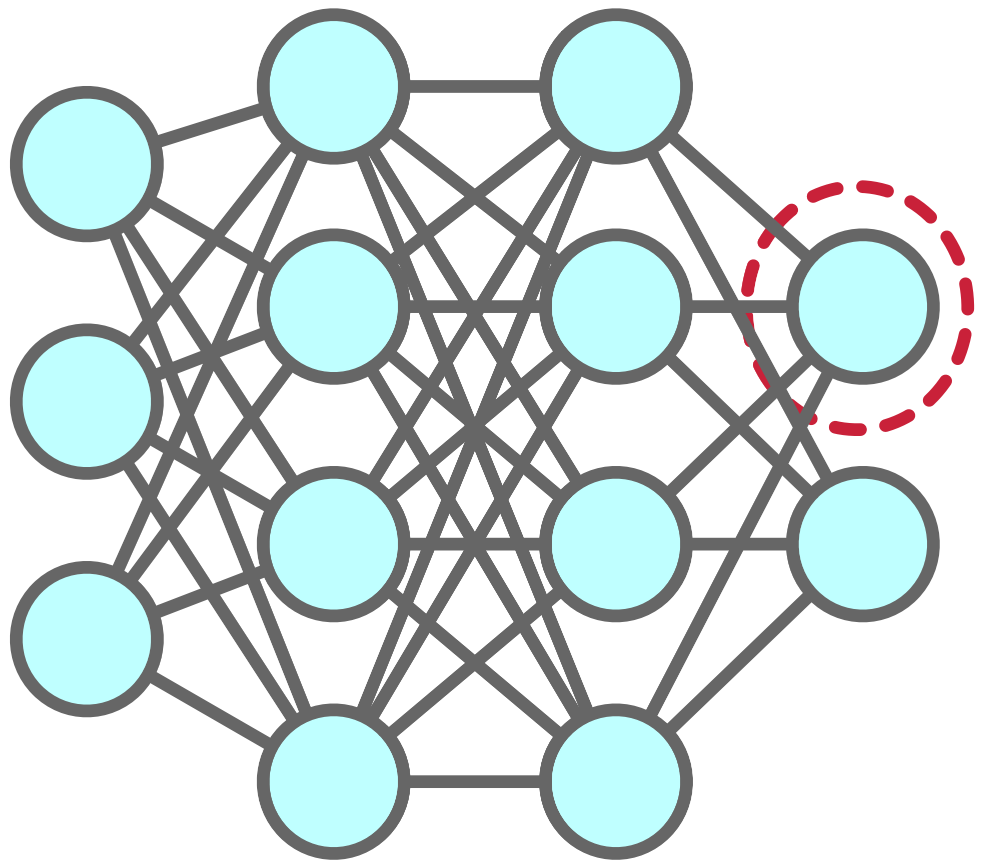
Processing in memory

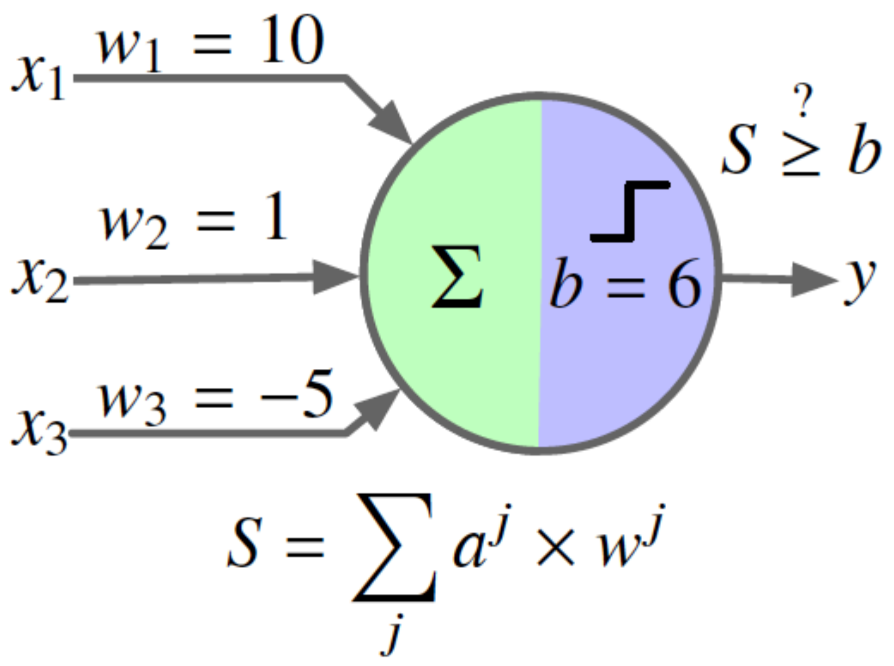


Perform computation **in the memory** without fetching data to the processor

A New Approach: Logic-Based Neural Networks

Neurons are implemented as a ***Boolean logic circuit***

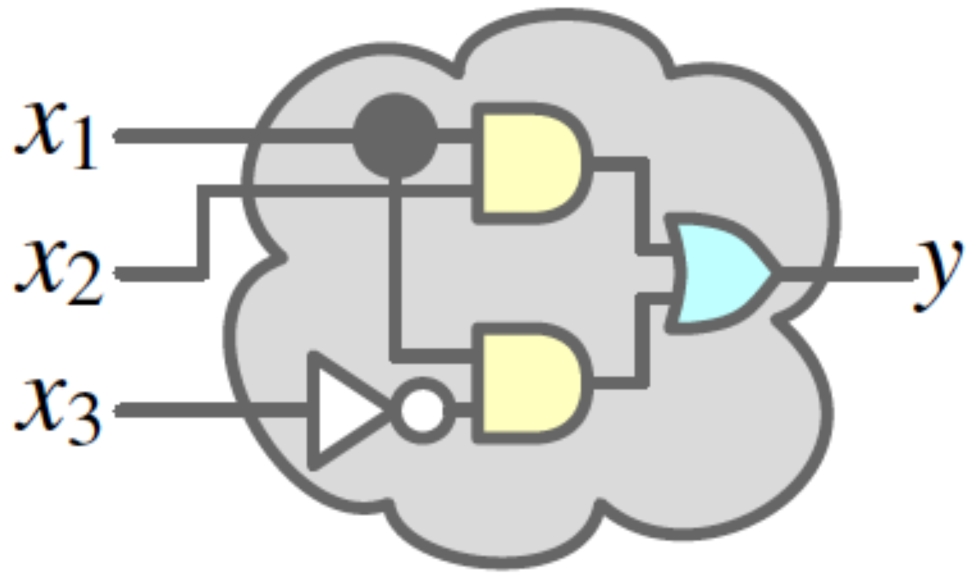




McCulloch-Pitts neuron

x_1	x_2	x_3	Σ	y
0	0	0	– 6	0
0	0	1	– 16	0
0	1	0	– 4	0
0	1	1	– 14	0
1	0	0	14	1
1	0	1	4	0
1	1	0	16	1
1	1	1	6	1

Truth table



$$y = x_1 x_2 + x_1 \overline{x_3}$$

Logic implementation

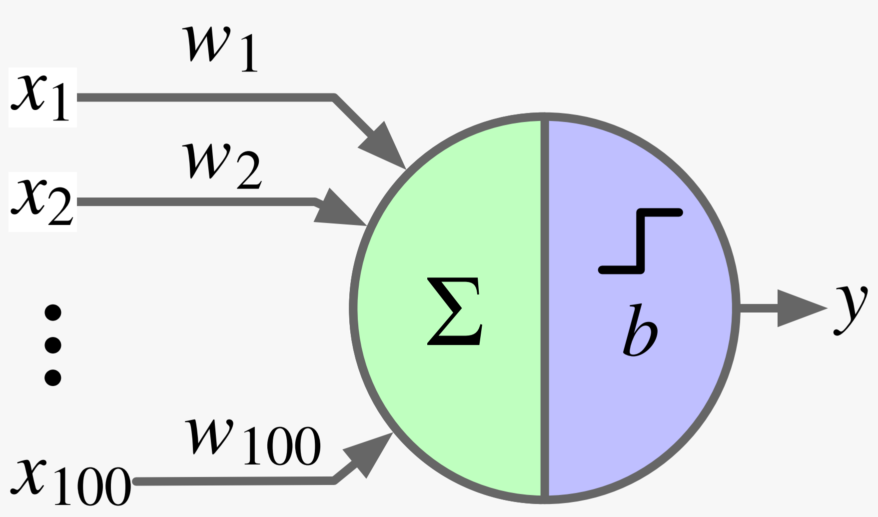
Pre-evaluated Boolean mappings functions of the neurons

Weights are **hardwired into the logic**, not stored in the memory

✓ No memory access overheads

[1] M. Nazemi *et al.*, "Energy-efficient, low-latency realization of neural networks through Boolean logic minimization," in *ASP-DAC*, 2019

Logic Size Reduction with Incompletely Specified Functions



100-input neuron

2^{100} input combinations

x_1	…	x_{97}	x_{98}	x_{99}	x_{100}	y
0	…	0	0	0	0	1
0	…	0	0	0	1	0
0	…	0	0	1	0	1
0	…	0	0	1	1	1
0	…	0	1	0	0	1
0	…	0	1	0	1	0
0	…	0	1	1	0	1
0	…	0	1	1	1	0
0	…	1	0	0	0	1
0	…	1	0	0	1	0
0	…	1	0	1	0	0
0	…	1	0	1	1	0
0	…	1	1	0	0	1
0	…	1	1	0	1	1
0	…	1	1	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	…	1	1	1	1	0

Completely specified function (CSF)

Not feasible

Input	x_1	…	x_{97}	x_{98}	x_{99}	x_{100}	y
	0	…	0	0	0	0	X
	0	…	0	0	0	1	X
	0	…	0	0	1	0	X
	0	…	0	0	1	1	X
	0	…	0	1	0	0	X
	0	…	0	1	0	1	X
	0	…	0	1	1	0	1
	0	…	0	1	1	1	X
	0	…	1	0	0	0	X
	0	…	1	0	0	1	X
	0	…	1	0	1	0	0
	0	…	1	0	1	1	X
	0	…	1	1	0	0	X
	0	…	1	1	0	1	X
	0	…	1	1	1	0	X
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	1	…	1	1	1	1	X

Incompletely specified function (ISF)
only for **observed inputs**

Smaller logic

[1] M. Nazemi *et al.*, "Energy-efficient, low-latency realization of neural networks through Boolean logic minimization," in *ASP-DAC*, 2019

Motivation: Unexploited Error Resilience

Input	x_1	…	x_{97}	x_{98}	x_{99}	x_{100}	y
	0	…	0	0	0	0	X
	0	…	0	0	0	1	X
	0	…	0	0	1	0	X
	0	…	0	0	1	1	X
	0	…	0	1	0	0	X
	0	…	0	1	0	1	X
	0	…	0	1	1	0	1
	0	…	0	1	1	1	X
	0	…	1	0	0	0	X
	0	…	1	0	0	1	X
	0	…	1	0	1	0	0
	0	…	1	0	1	1	X
	0	…	1	1	0	0	X
	0	…	1	1	0	1	X
	0	…	1	1	1	0	X
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	1	…	1	1	1	1	X

Observed for many inputs ➡

Observed only for 1 input ➡

⬅ **Important for accuracy**

⬅ **Not important for accuracy**

Not all input combinations are equally important

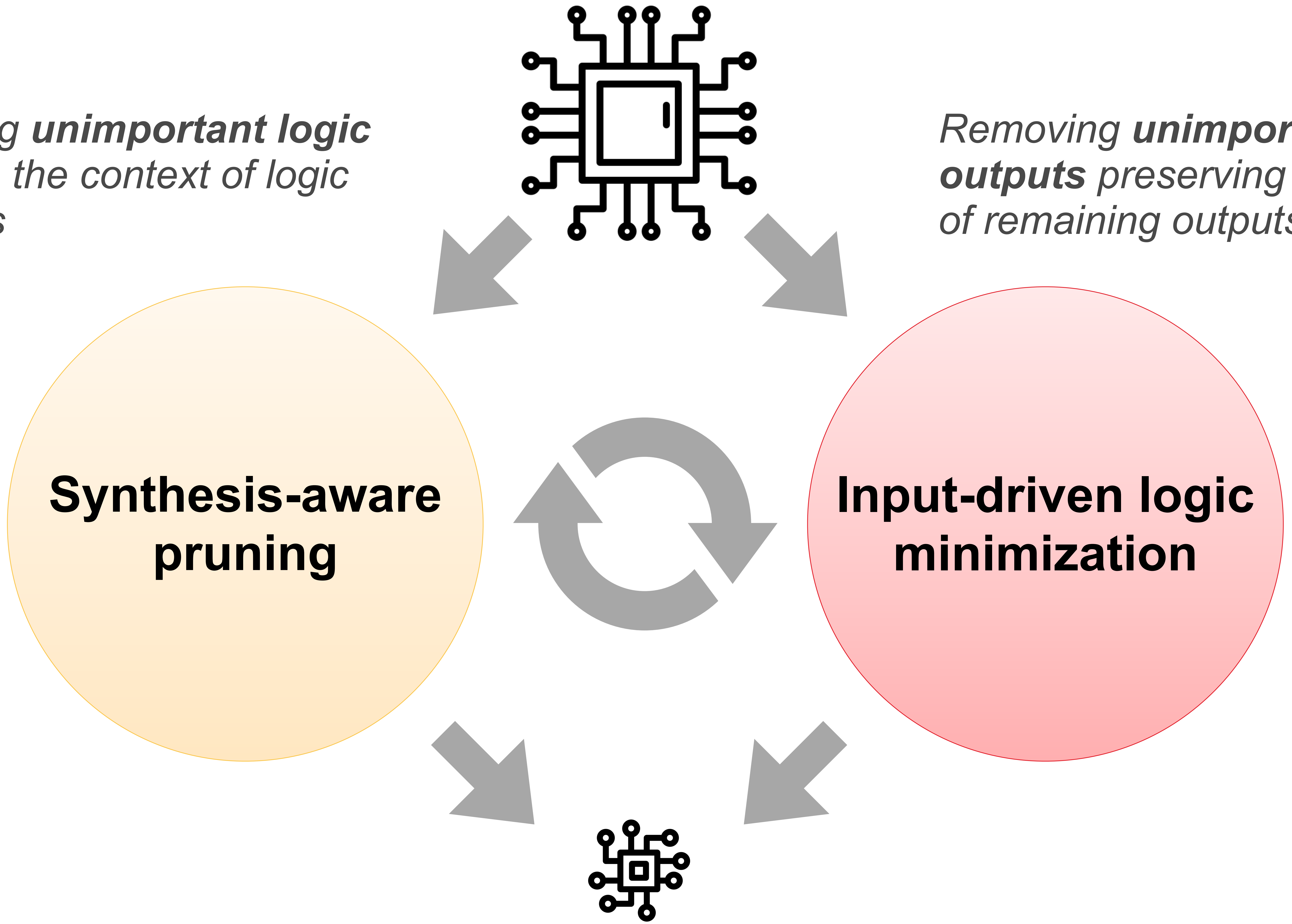
= Opportunity for further logic reduction without accuracy degradation

Question: How do we identify and remove unimportant outputs?

Proposed Method: Two-coordinate Logic Minimization

*Removing **unimportant logic inputs** in the context of logic synthesis*

*Removing **unimportant logic outputs** preserving importance of remaining outputs*



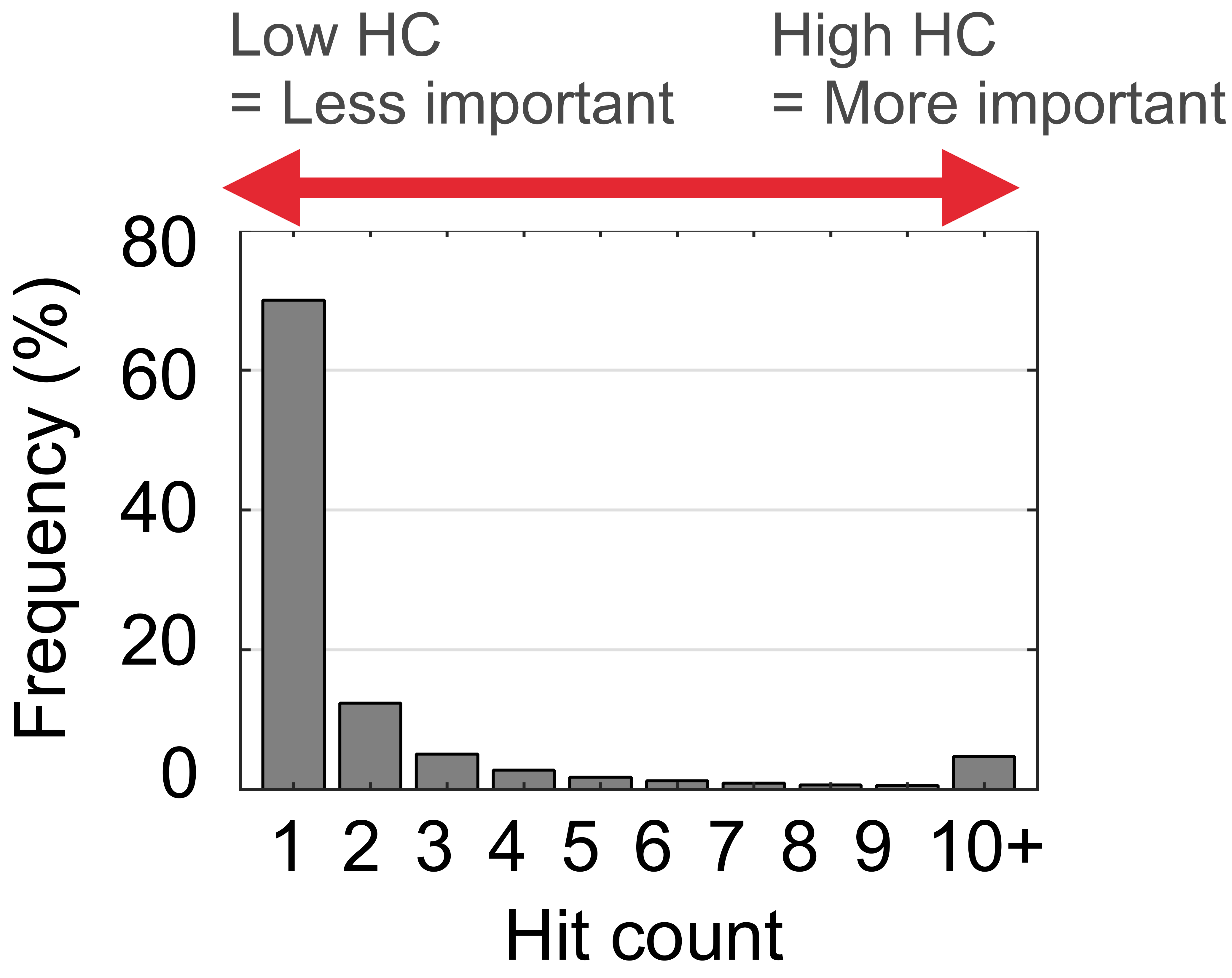
Hitcount: An Importance Metric of Inputs

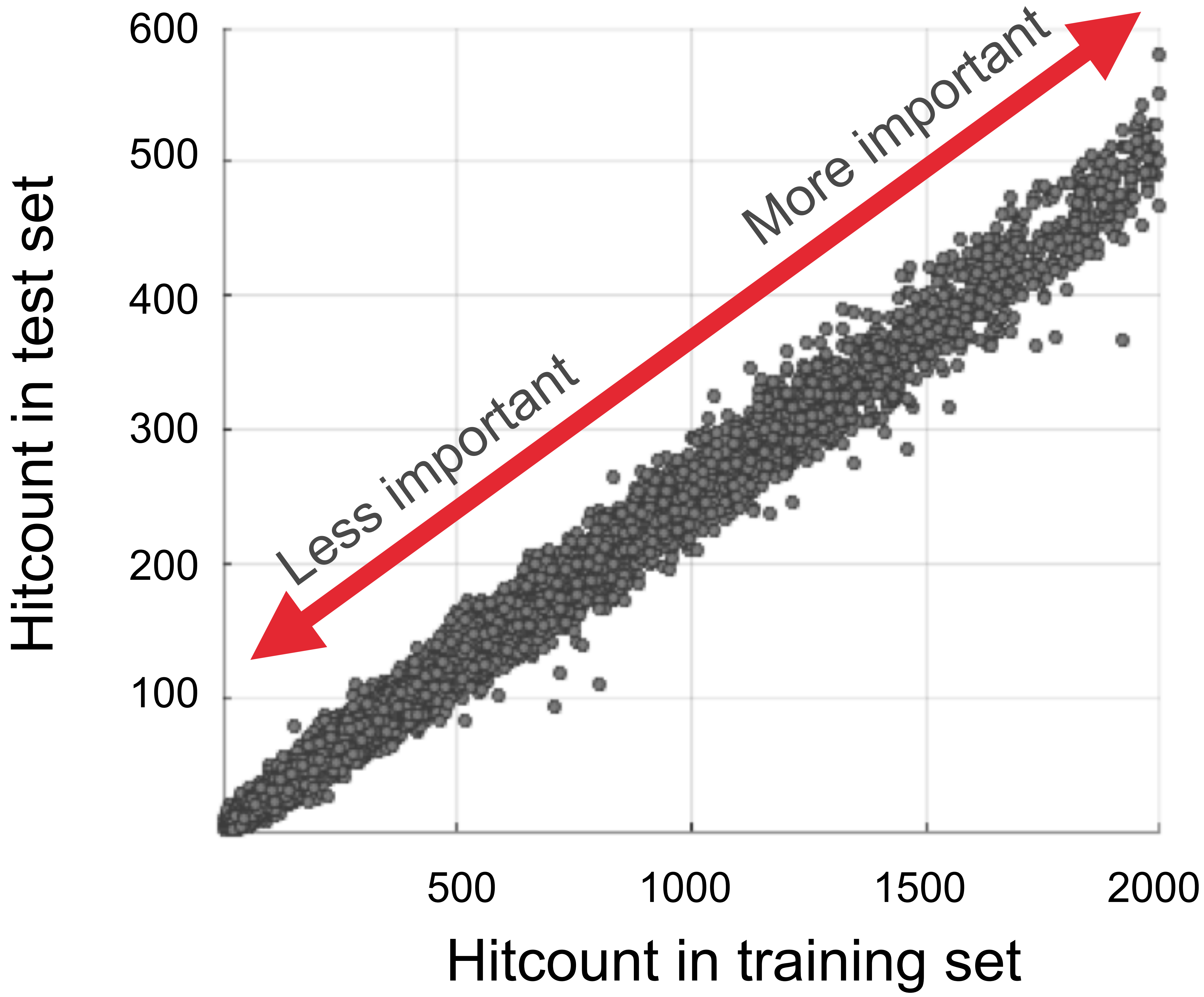
- **Hitcount (HC):** Number of encounters of input combination
- High HC
  = High input encounter frequency
  = Important input combination
- Output is specified only when the input is encountered at least once during training (i.e., HC ≥ 1)

Input	x_1	…	x_{97}	x_{98}	x_{99}	x_{100}	y	HC
	0	…	0	0	0	0	X	**0**
	0	…	0	0	0	1	X	**0**
	0	…	0	0	1	0	X	**0**
	0	…	0	0	1	1	X	**0**
	0	…	0	1	0	0	X	**0**
	0	…	0	1	0	1	X	**0**
	0	…	0	1	1	0	1	**10000**
	0	…	0	1	1	1	X	**0**
	0	…	1	0	0	0	X	**0**
	0	…	1	0	0	1	X	**0**
	0	…	1	0	1	0	0	**1**
	0	…	1	0	1	1	X	**0**
	0	…	1	1	0	0	X	**0**
	0	…	1	1	0	1	X	**0**
	0	…	1	1	1	0	X	**0**
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	**0**
	1	…	1	1	1	1	X	**0**

HITCOUNT: UNEQUAL IMPORTANCE OF INPUTS

Dataset: MNIST





Only a very small subset of input combinations are important

HC in the traning set well represents the importance of input combinations in the test set

METHOD 1: SYNTHESIS-AWARE PRUNING

- Magnitude-based pruning – Eliminate inputs with low magnitudes
- **Update HC:** Add HCs of merged rows $$HC' = \sum_{i \in I} HC_i$$

x_1	x_2	x_3	y	HC
10	1	-5		
0	0	0	0	2000
0	0	1	X	0
0	1	0	X	0
0	1	1	X	0
1	0	0	1	5
1	0	1	0	10
1	1	0	1	150
1	1	1	1	1000

Remove low-magnitude input x_2

x_1	x_2	x_3	y	HC
10	1	-5		
0	X	0	0	2000
0	X	1	X	0
0	X	0	X	0
0	X	1	X	0
1	X	0	1	5
1	X	1	0	10
1	X	0	1	150
1	X	1	1	1000

x_1	x_3	y'	HC'	
10	-5			
0	0	?	2000+0=	2000
0	1	?	0+0=	0
1	0	?	5+150=	155
1	1	?	10+1000=	1010

Merge rows and update HC

Method 1: Synthesis-aware Pruning

- **Update output:** Weighted average of merged rows

$$y' = \mathrm{round}\left(\frac{\sum_{i \in I} y_i \times HC_i}{\sum_{i \in I} HC_i}\right)$$

x_1	x_2	x_3	y	HC
10	1	-5		
0	X	0	0	2000
0	X	1	X	0
0	X	0	X	0
0	X	1	X	0
1	X	0	1	5
1	X	1	0	10
1	X	0	1	150
1	X	1	1	1000

x_1	x_3	y'		HC'
10	-5			
0	0	round((0×2000+0×0)/2000)=	0	2000
0	1	round((X×0+X×0)/0)=	X	0
1	0	round((1×5+1×150)/155)=	1	155
1	1	round((0×10+1×1000)/1010)=	1	1010

Update output

Method 2: Input-driven Logic Minimization

- Unspecify output (switch to Don't-Care) for unimportant inputs
- Makes the truth table even more sparse, resulting in even smaller logic
- HC remains unchanged

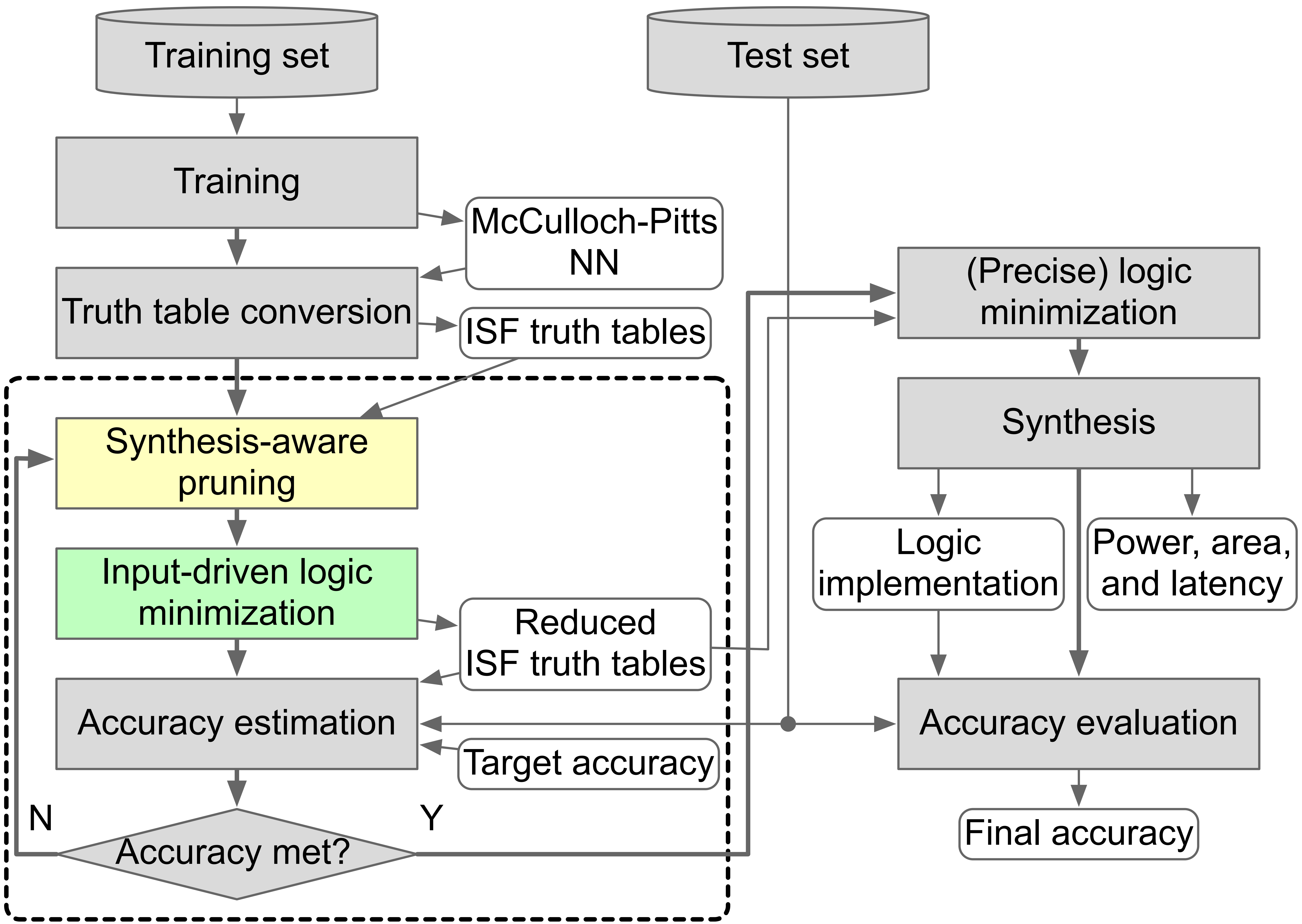
x_1	x_2	x_3	y	HC
10	1	-5		
0	0	0	0	2000
0	0	1	X	0
0	1	0	X	0
0	1	1	X	0
1	0	0	1	**5**
1	0	1	0	**10**
1	1	0	1	**150**
1	1	1	1	**1000**

$\left.\right\}$ Low HC

x_1	x_2	x_3	y'	HC
10	1	-5		
0	0	0	0	2000
0	0	1	X	0
0	1	0	X	0
0	1	1	X	0
1	0	0	**X**	5
1	0	1	**X**	10
1	1	0	**X**	150
1	1	1	**X**	1000

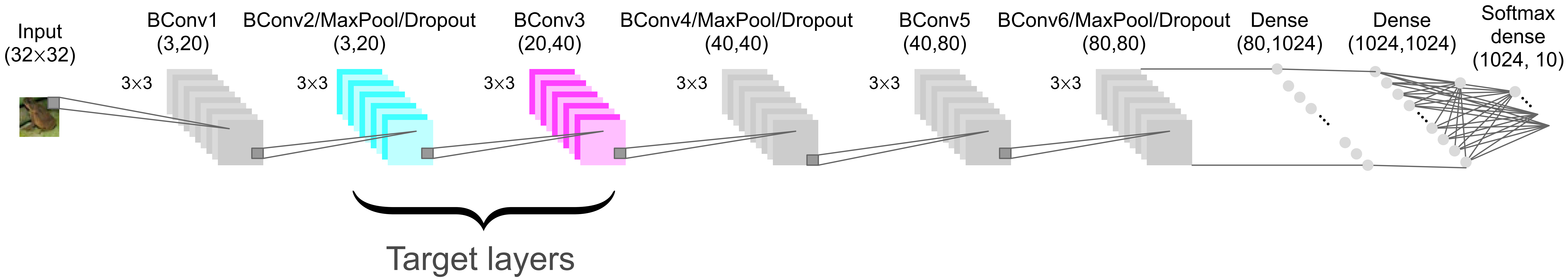
↑ Output unspecified

Two Methods Applied in Design Flow



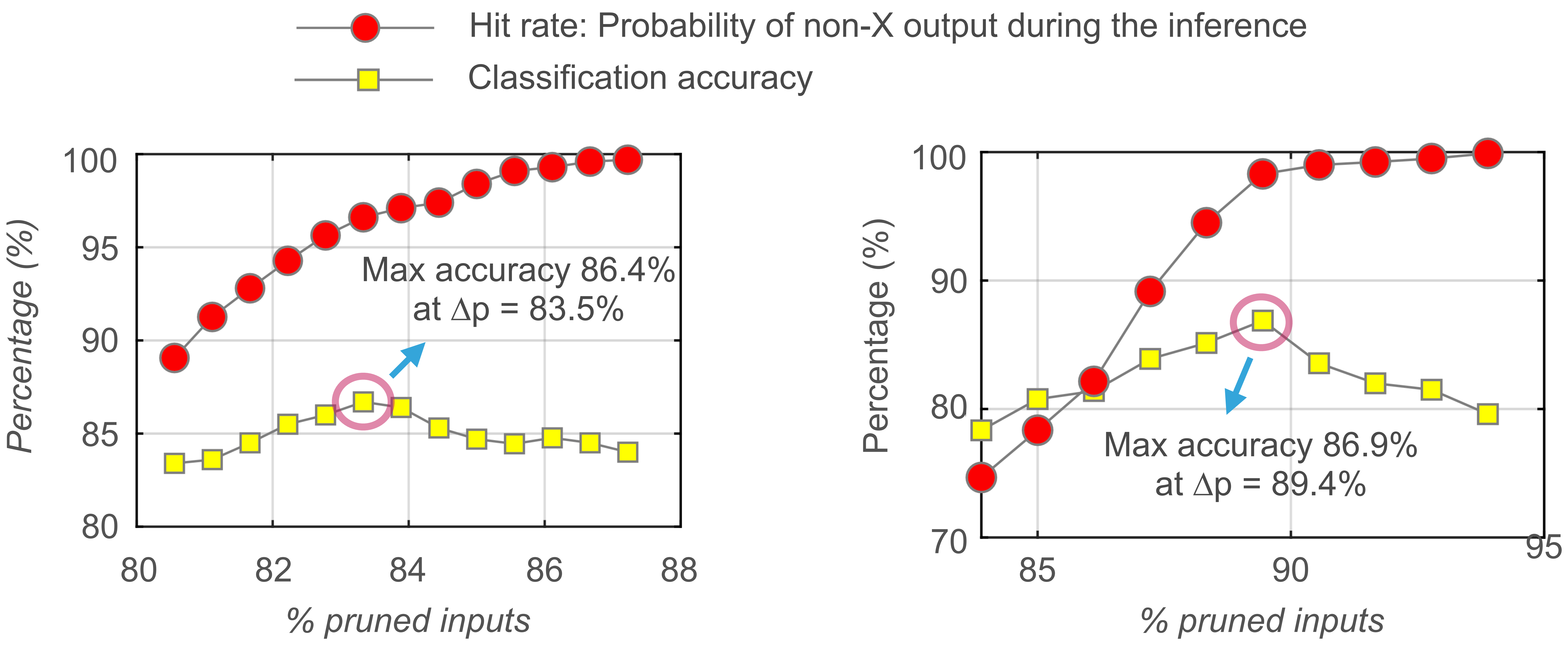
- Synthesis-aware pruning and input-driven logic minimization iteratively applied
- Accuracy estimated from reduced truth tables without synthesis
- Repeated until target accuracy is met

EXPERIMENTAL SETUP



- Model: A CNN with binarized activation convolutional layers
- Target layers: 2nd and 3rd conv layers
- Dataset: CIFAR-10
- Validation: Last 5000 samples of training set as validation set after 200 epochs

Results: Synthesis-aware Pruning

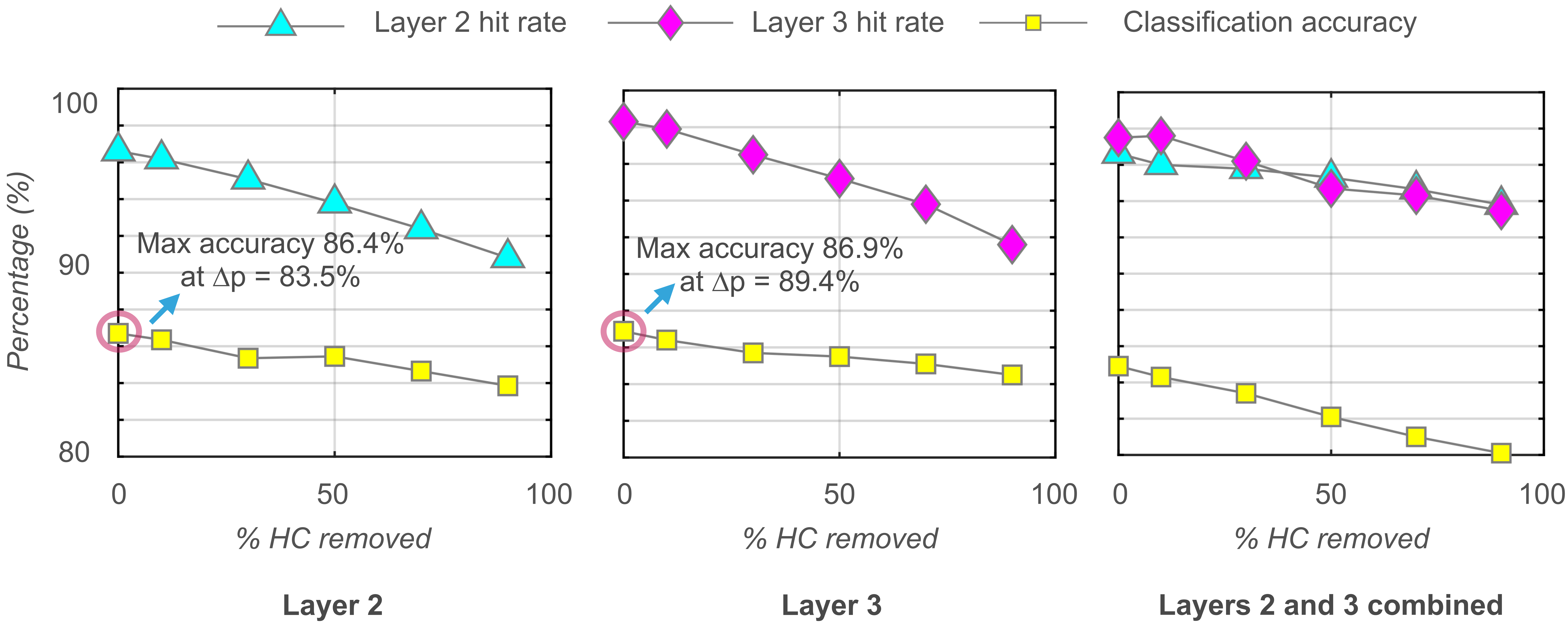


Layer 2

Layer 3

80%–90% inputs can be pruned, resulting in increased hit rate and accuracy

RESULTS: INPUT-DRIVEN LOGIC MINIMIZATION



High accuracy maintained even with ~90% HC removed

Hardware Evaluation

Layer	CLNN (Proposed)			SCALE-Sim energy per image (nJ)	Energy reduction per image
	Energy per cycle (pJ)	Cycles per image	Energy per image (nJ)		
2	541	1,024	553	5,682	**90.3%**
3	249	256	64	11,360	**99.4%**

- Synthesis: Synopsys Design Compiler using TSMC 45 nm libarary
- Baseline: Systolic array simulated using SCALE-Sim

90+% energy reduction per image

[1] A. Samajdaret al., "SCALE-Sim: Systolic CNN accelerator simulator,"arXiv preprint arXiv:1811.02883, 2018

Conclusion

- Identified **unexploited energy-saving opportunities** in combinational logic neural network implementation
  - Introduced **hitcount** as a unique importance metric of inputs in combinational logic neural network
- Proposed **two methods for reducing model size** in the context of logic synthesis
  - **Synthesis-aware pruning** and **input-driven logic minimization**
  - Exploits **error resilience** of neural networks to further reduce logic size and enhance scalability
- **90+% power reduction** while maintaining a competitive accuracy of **82%** on the CIFAR-10 dataset