UCLA Samueli School of Engineering

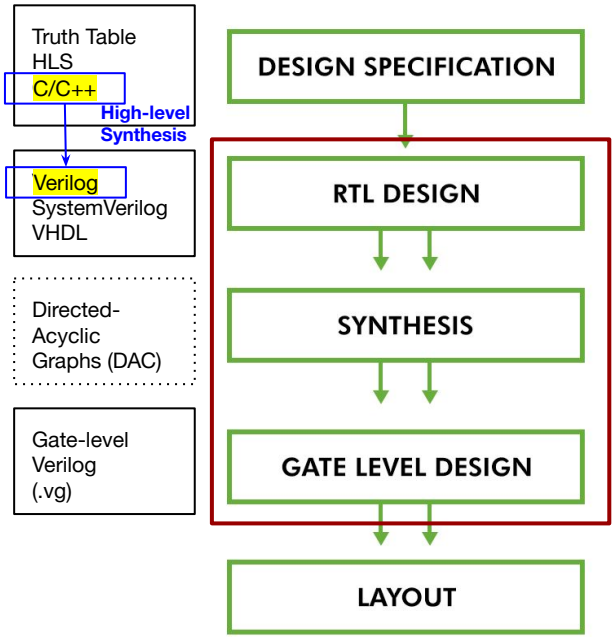
Machine Learning for Logic Synthesis

CS259 Class Presentation T10
February 29, 2024

Andrew Fantino, Parangat Mittal
Dept. of Electrical and Computer Engineering

Idea to Chip

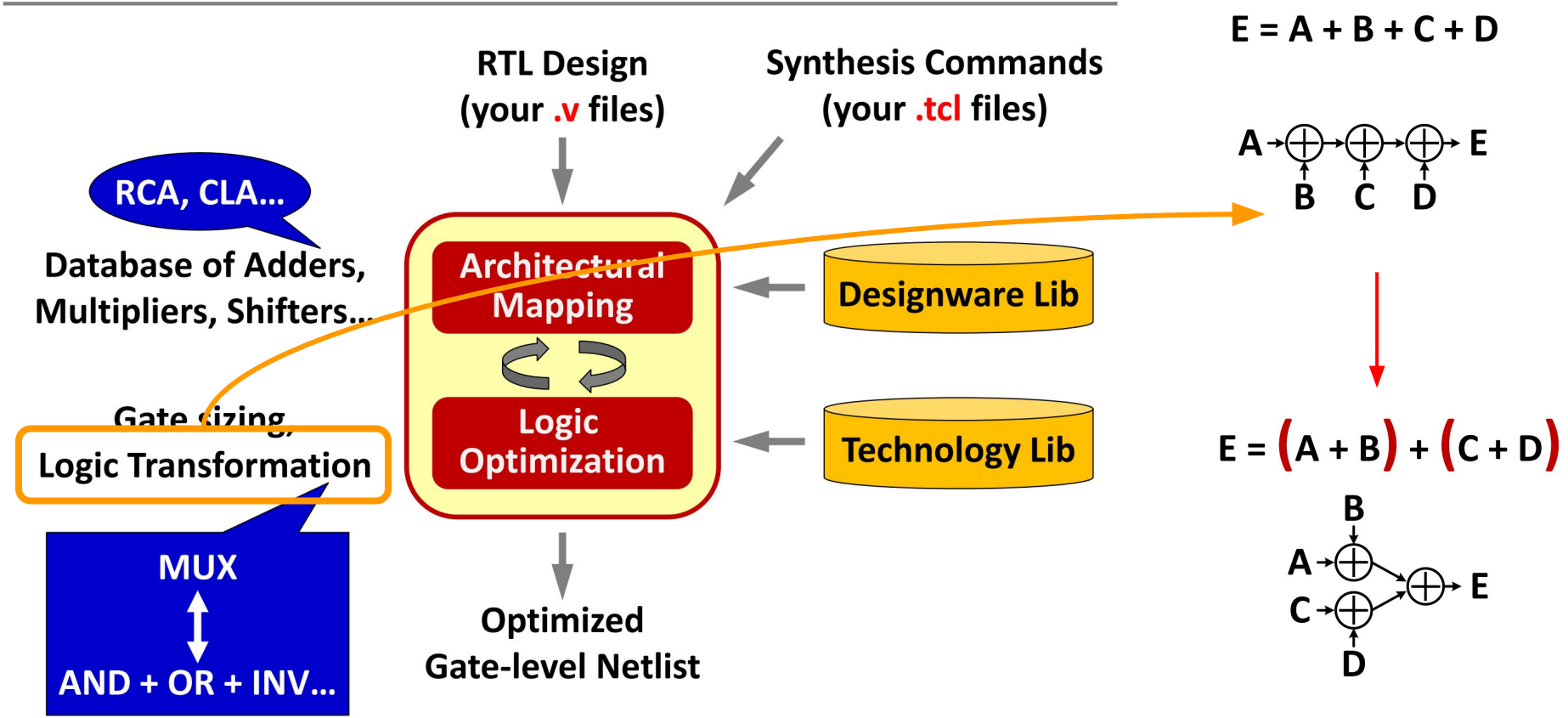
Conversion of a high-level design representation to an equivalent optimized gate-level representation specific to a technology is **Synthesis**



- Series of transformations performed to the design to optimize it in terms of power and timing.
- Represented as Directed-Acyclic Graph (DAG) as an intermediate form.
- Optimized DAG representation yields a good design.

- Advanced technology nodes increase the search space for exploration with intensive iterations.
- Finding the best-fit synthesis flow from the entire search space is time-consuming and impossible to completely cover.
- Need to find efficient solution to:
  - Find the best **synthesis flow**
  - Optimize the **DAG representation**

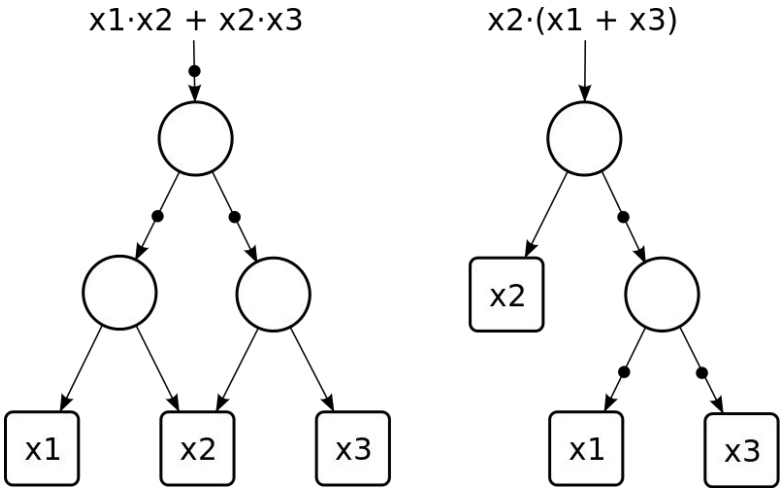
How to do Synthesis?



Logic Representation as a Graph

And Inverter Graph
Uses only AND and NOT gates to represent (Universal NAND gate)

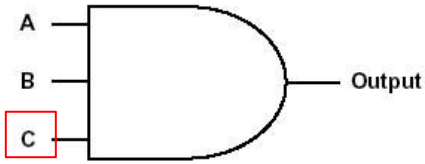
- Structural representation of a logical implementation
- Two-input nodes for conjunction
- Terminal nodes labeled with variable names
- Edges containing markers for logical negation.



Logic Representation as a Graph

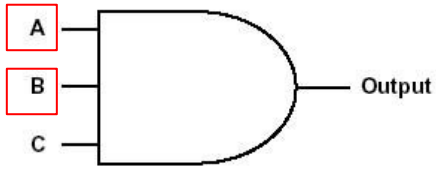
Majority-of-3 gate

If A=B, output = A



Not dependent on C

If A≠B, output = C

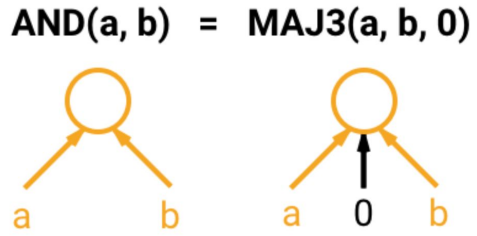


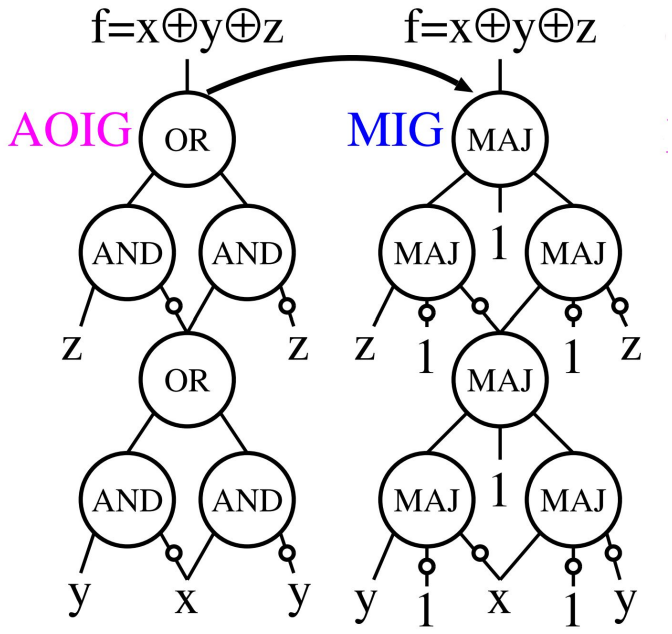
Not dependent on A/B

Majority Inverter Graph

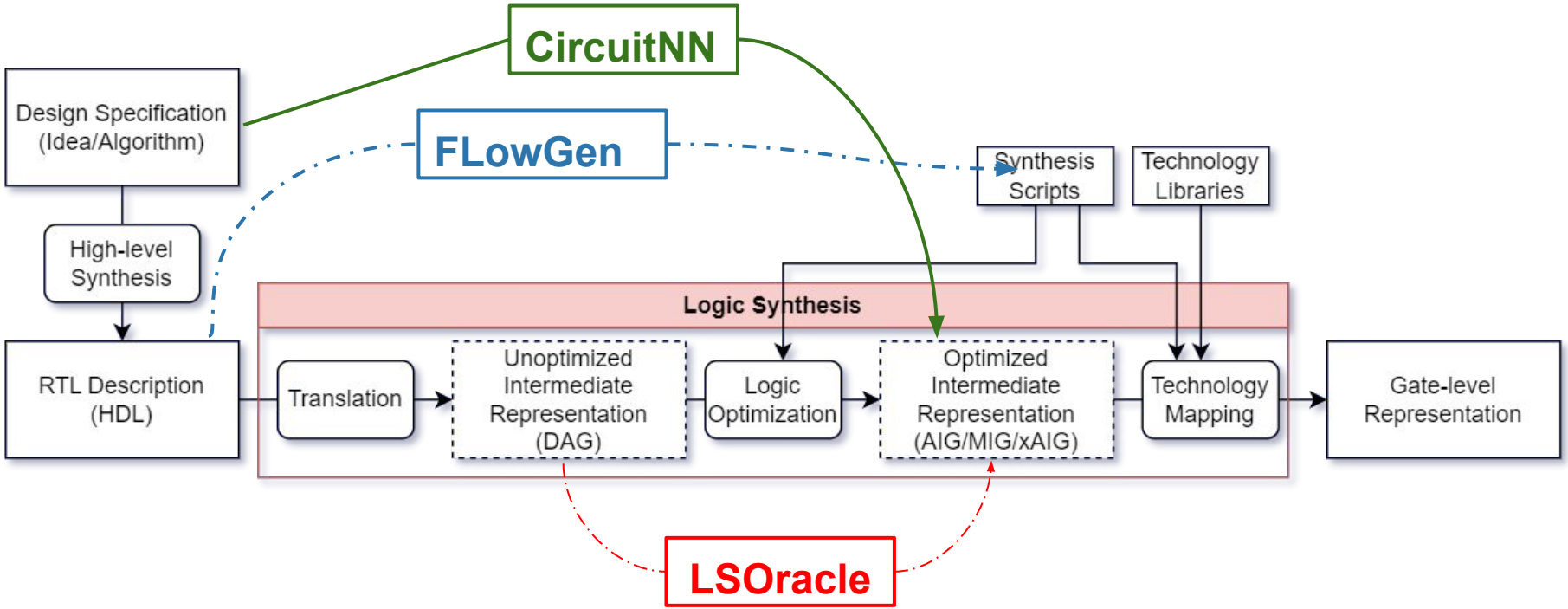
Uses only Majority and NOT gates to represent

Encodes in equal number of nodes as AIG, and even more by appropriate transformations.



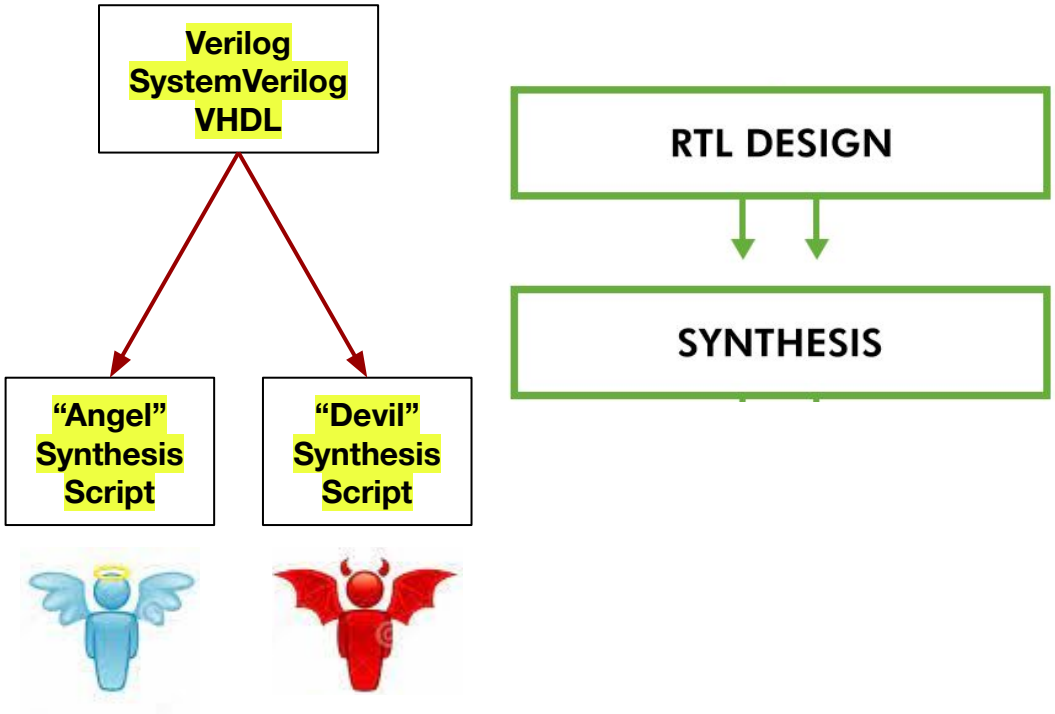


Birds Eye View



Developing Synthesis Flows without Human Knowledge

Cunxi Yu, Houping Xiao,
Giovanni De Micheli



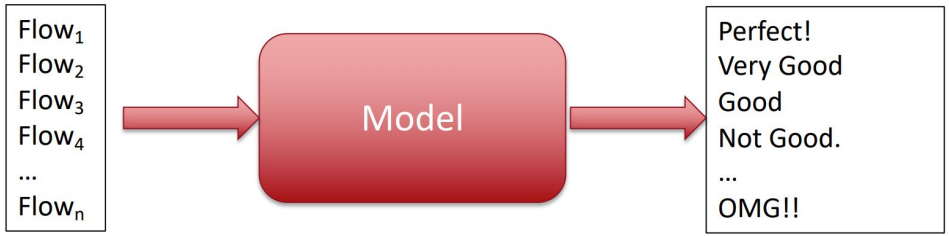
Problem Formulation

Order matters!

Problem:
Choosing the best sequence of transformations on the design (**Synthesis Flow**) is important to get the best QoR.

Testing each design flow from the large search space is impractical and close to impossible. (6 instr types, 4 operations: 3.15e+15 outcomes)

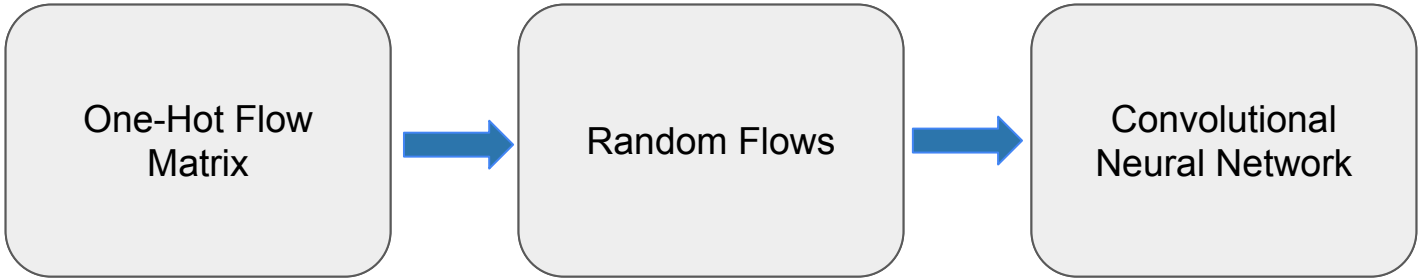
Solution:
Classify synthesis flows as good and bad?
Predict QoR of the design under that flow?



FlowGen: Overview

For any classification problem, we need:

1. **Input Representation**: Define a suitable representation for input data.
2. **Training Dataset**: Gather a labeled dataset for model training.
3. **Classification Model**: Deep Neural Network to classify the represented input-data into one of the output classes/labels

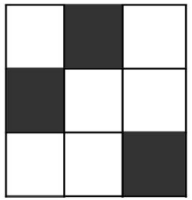


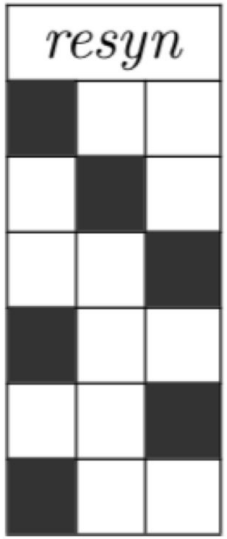
FlowGen: Input Representation

Synthesis flows are represented as one-hot binary matrices

- Example: balance (b), rewrite (rw), rw -z (rwz)
  - b = [1 0 0]
  - rw = [0 1 0]
  - rwz= [0 0 1]
- Flow: rw -> b -> rwz = [rw;b;rwz]

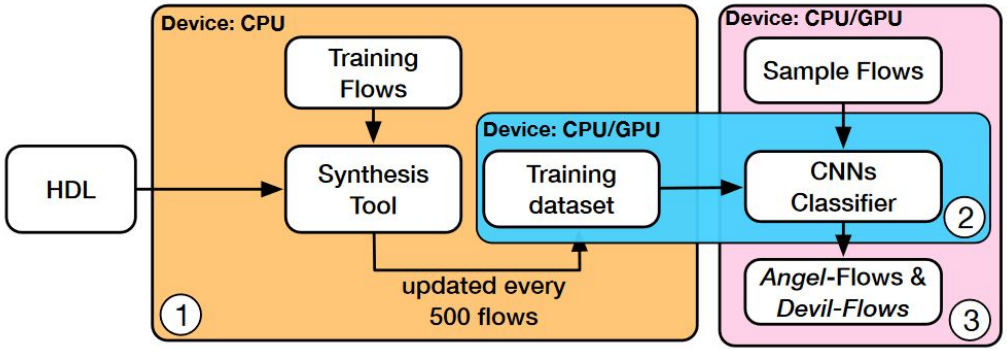
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





FlowGen: Training Dataset

1. Run random synthesis flow on a design
   a. Takes a long time!!!
2. Evaluate QoR and label it a class
   a. Eg: {5%, 15%, 40%, 65%, 90%, 95%} buckets
3. Repeat 1000 times and start training model
4. Continue indefinitely, retraining model every 500 new samples



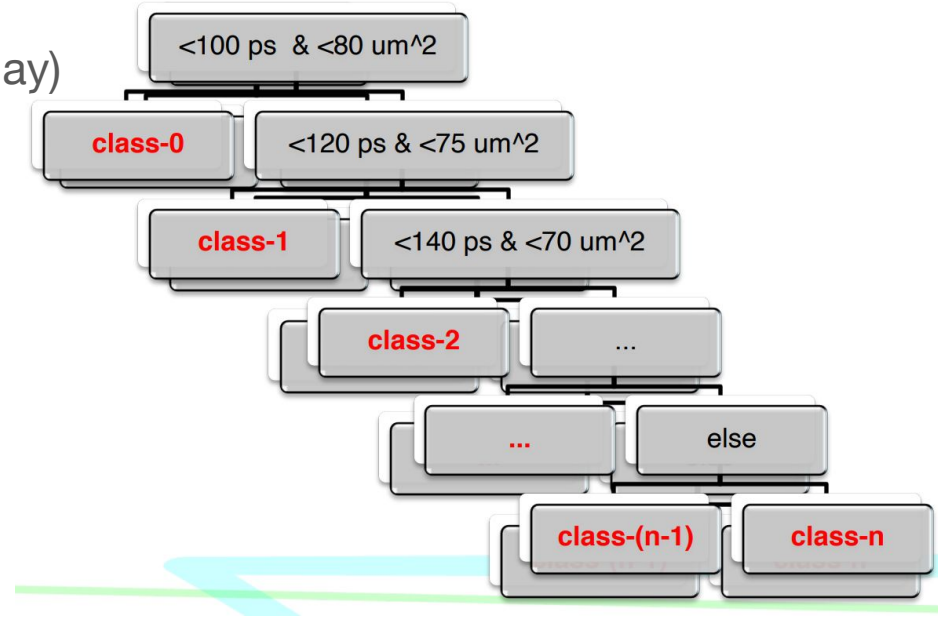
FlowGen: Labelling

0-n Labels/Classes are based on:

- Labelling Rule 1 - single metric (ex. delay)
- Labelling Rule 2 - multi-metric (ex. delay+area)

Class 0 - Best on the metric (Angel)
Class n - Worst on the metric (Devil)

Single-metric	Multi-metric	Class/Label
$r \leq x_0$	$r_0 \leq x_0, r_1 \leq y_0$	*0*
$x_0 < r \leq x_1$	$x_0 < r_0 \leq x_1, y_0 < r_1 \leq y_1$	*1*
$x_1 < r \leq x_2$	$x_1 < r_0 \leq x_2, y_1 < r_2 \leq y_2$	*2*
...	...	...
$r > x_n$	$r_0 > x_n, r_1 > y_n$	*n*

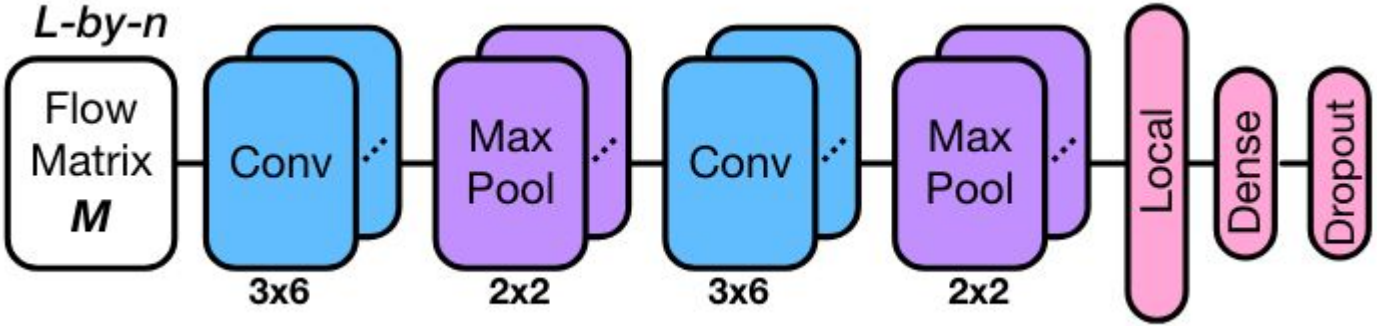


FlowGen: Who Cares About the Devil?

"*devil-flows* could provide information for improving the synthesis transformations."

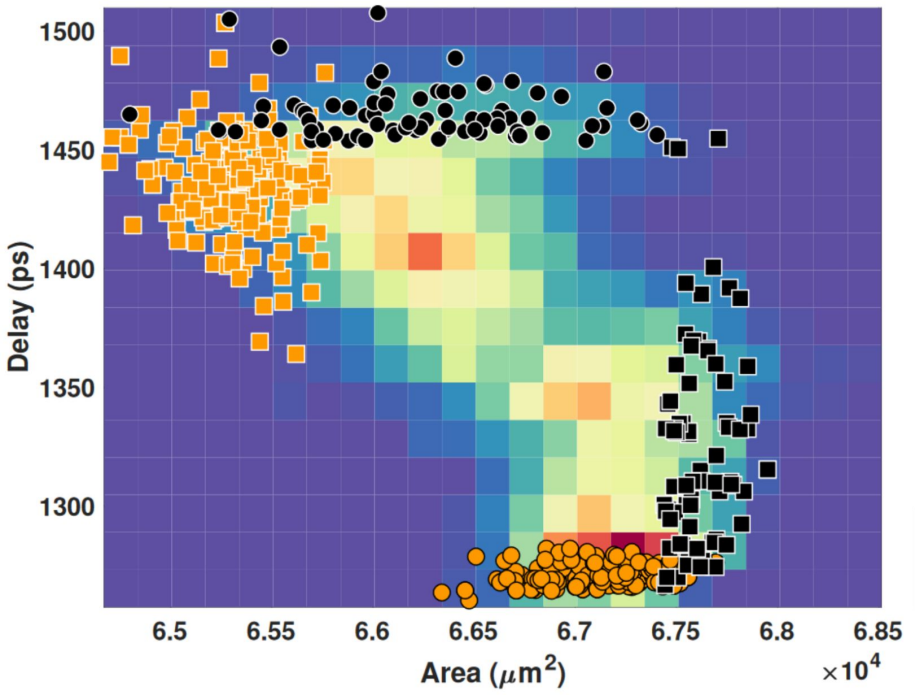
FlowGen: Classification Model

- Input Shape (1, 24, 6) -> Output Shape (1, 7)
- Classify likelihood of synthesis script falling into 1 of the 7 QoR buckets.
- Take the lowest and highest QoR bucket as your devil and angel flows

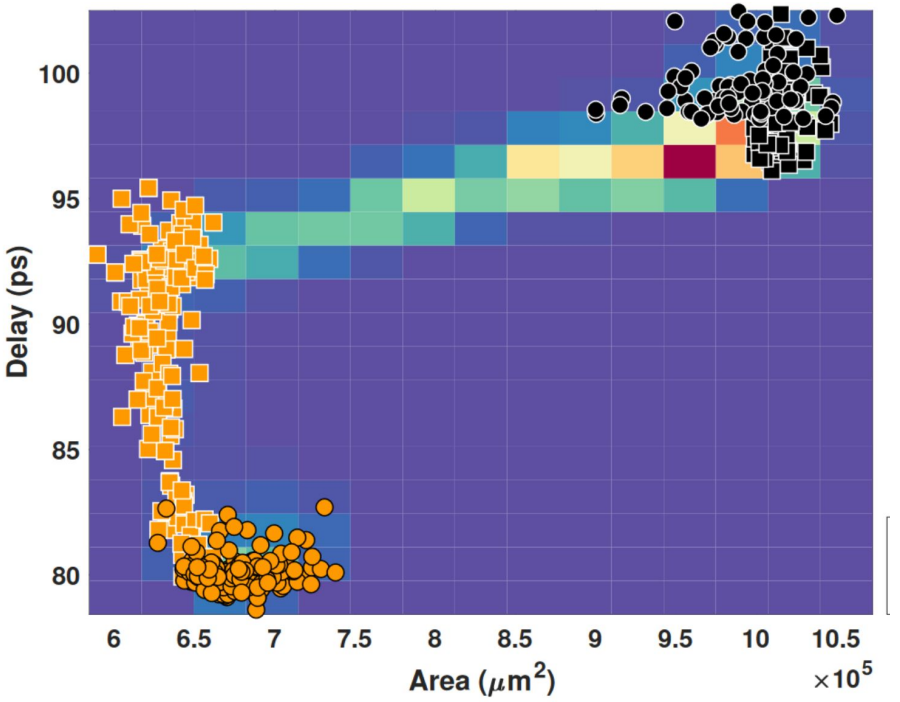


**Note: Demo code does not match structure of model here & model from paper does not provide results expected

FlowGen: Results



64-bit Montgomery Multiplier



128-bit AES Core

FlowGen: Limitations

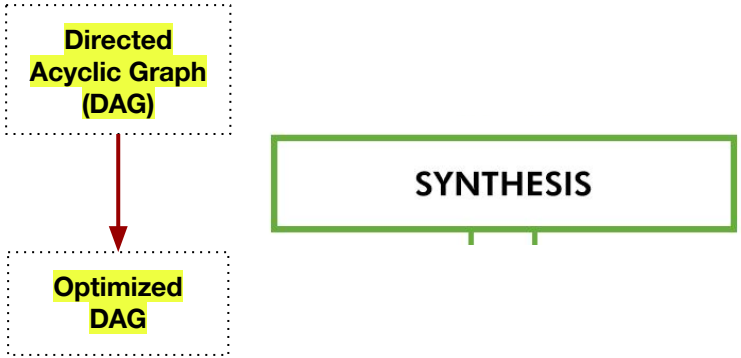
- High runtime - 3-4 days
  - Classifying each unknown flow into a class
  - On-the-fly training, so even though lesser time than actually running each flow, still a lot of time to cover the entire search space

- 1 Design per Model
  - A model trained on a design, cannot be used for inference under a new design
  - Each design responds differently to same synthesis flow

- Poor Scaling
  - One-hot inputs makes more complex synthesis scripts grow dramatically

LSOracle: a Logic Synthesis Framework Driven by Artificial Intelligence

Walter Lau Neto, Max Austin, Scott Temple, Luca Amaru, Xifan Tang, Pierre-Emmanuel Gaillardon



Motivation

Problem:

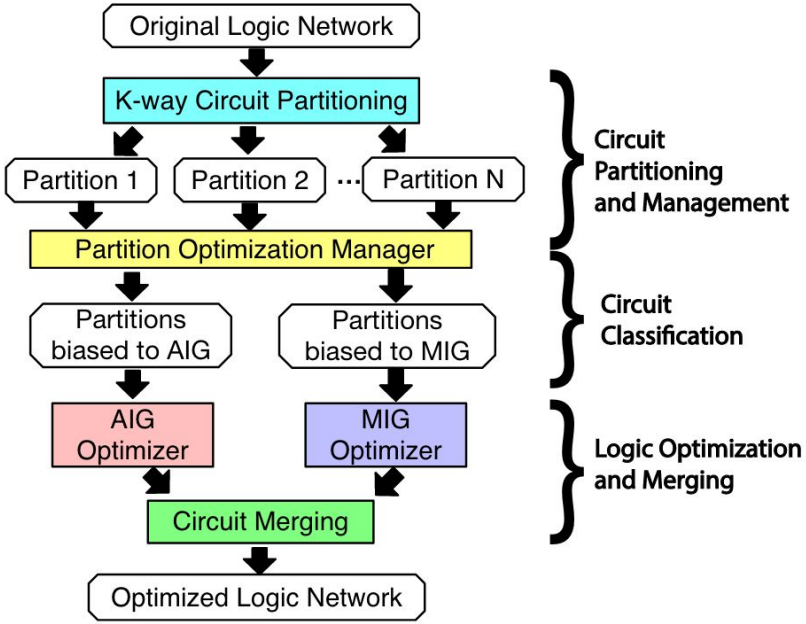
- AIGs and MIGs are good logic optimizers for different types of logic blocks.
- When different functional blocks co-exist, using only ONE optimizer on each block is not the complete approach to get the best results.
  - AIG - delivers smallest circuits
  - MIG - delivers fastest circuits

Solution:

- Use AIGs or MIGs both on the different blocks or partitions of the circuit
  - **Compare** metrics for both and choose the better one?
  - **Predict** the best-fit optimizer for each portion of the design?

LSOracle: Overview

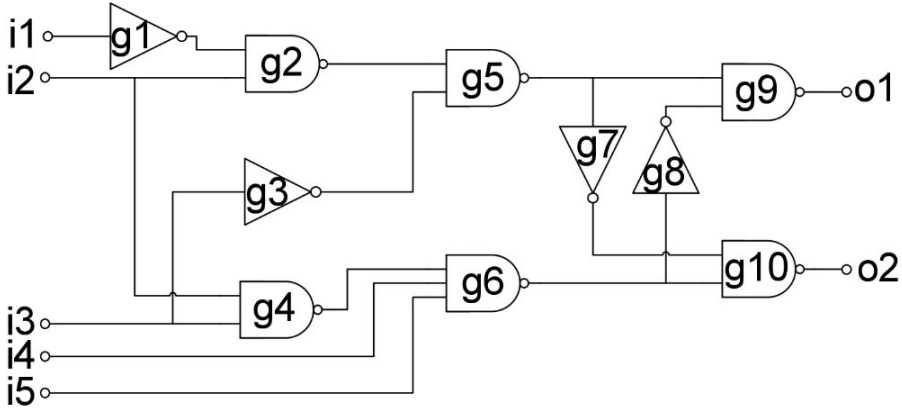
1. **Split** the circuit into k-way partitions
2. **Classify** each partition as AIG or MIG
3. **Optimize** the partition
4. **Merge** the partitions back

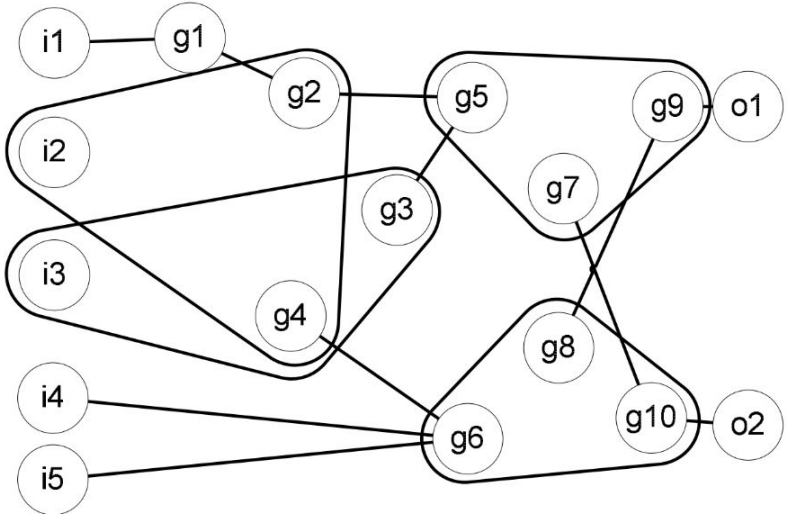


LSOracle: Circuit Partitioning

Multi-level k-way hypergraph partitioning problem

- Hypergraph representation for input DAG partitioned via ***KaHyPar***
- All inter-partition connections stored to be used at the end for merging

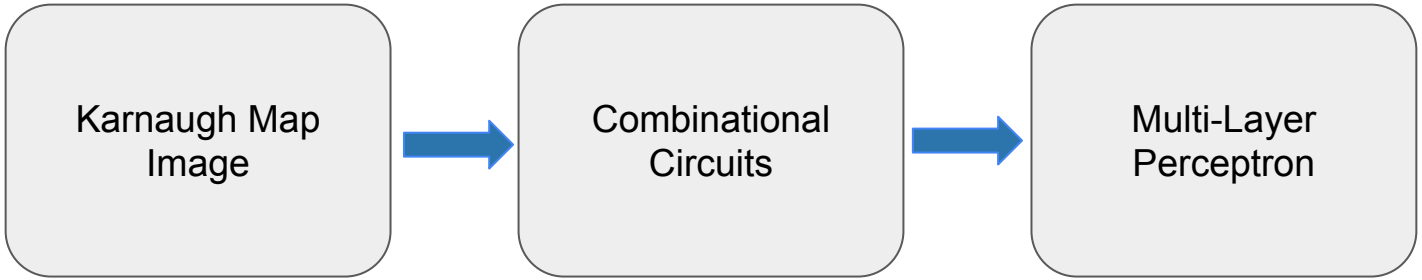




LSOracle: Circuit Classification

For any classification problem, we need:

1. **Input Representation**: Define a suitable representation for input data.
2. **Training Dataset**: Gather a labeled dataset for model training.
3. **Classification Model**: Deep Neural Network to classify the represented input-data into one of the output classes/labels

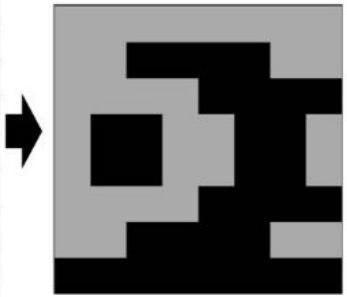


LSOracle: Circuit Classification

Input Representation - KMI (Karnaugh Map Images)

- n-input boolean function can projected in 2D space using K-maps
- Handy method to manually simplify boolean expressions
- Using Karnaugh-map representation of boolean functions as images

ABC \ DEF	000	001	011	010	110	111	101	100
000	1	1	1	1	1	1	1	1
001	1	1	0	0	0	0	1	1
011	1	1	1	1	0	0	0	0
010	1	0	0	1	1	0	0	1
110	1	0	0	1	1	0	0	1
111	1	1	1	1	0	0	0	0
101	1	1	0	0	0	0	1	1
100	0	0	0	0	0	0	0	0

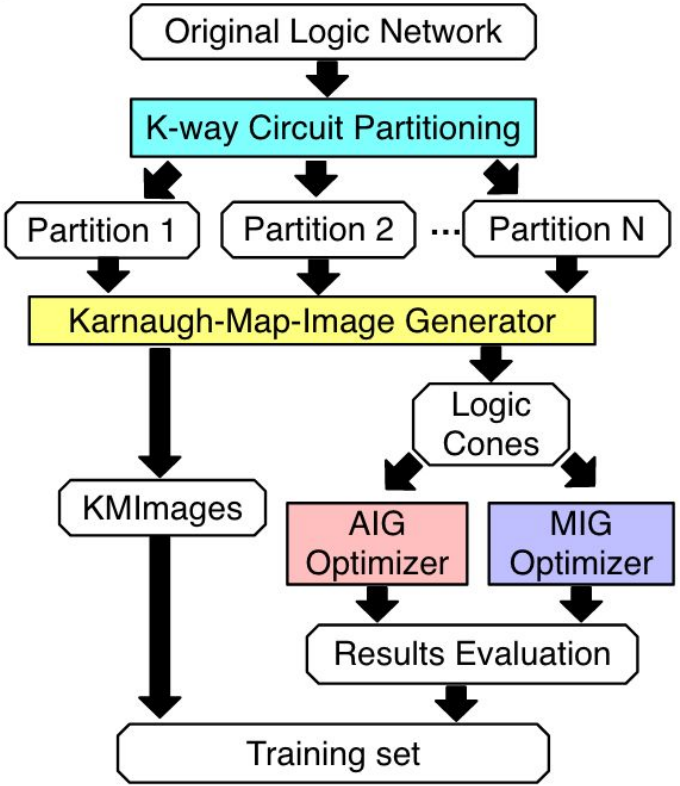


KMImage

LSOracle: Circuit Classification

Training Dataset - EPFL and ISCAS'89 benchmarks

- On the initial unlabelled functions -
  - Run k-way partitioning
  - Generate KMImage for each partition **(data)**
  - Run AIG and MIG optimizer on each partition
  - Evaluate results on least area (no of nodes)
  - Assign classes **(label)**

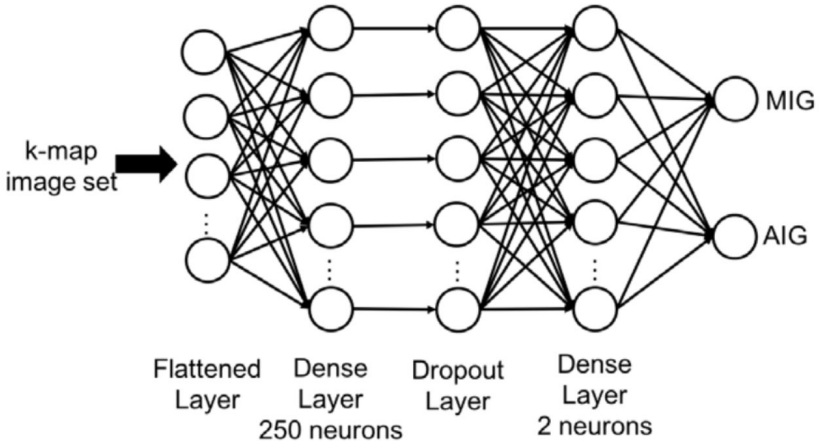


LSOracle: Circuit Classification

Classification Model - Deep Neural Network

Goal - Classify 2D black and white small image into one of the two classes

- Small search space of the problem
- **Multi-layer perceptron** with two dense layers separated by a dropout layer



LSOracle: Circuit Classification

Classification Algorithm -

Problem 1 - AIG and MIG both for a partition

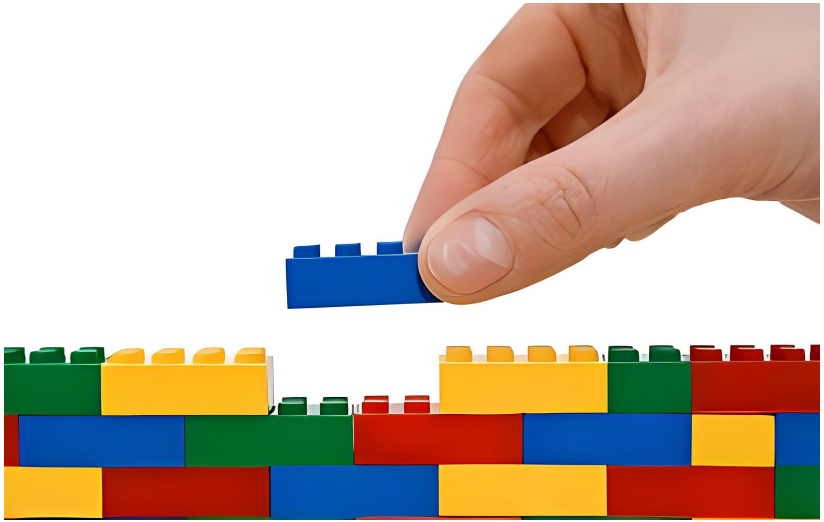
- KMIs of partitions with multiple outputs might be assigned to different optimizers.
- Give more weight to KMIs having a higher influence on the partition
  - More nodes and higher logic depth have greater impact on DAG
  - Weights assigned dynamically based on deviation from average

Problem 2 - > 16 inputs in a partition

- Classifier is designed to handle partitions with maximum 16 inputs
- Heuristic process is followed.
  - If logic depth of the partition is > 40% of total network depth, it is assigned to MIG
  - MIG optimizes potentially long critical paths better than AIG

LSOracle: Circuit Merging

- Each partition optimized by the chosen method
- Inputs and outputs set as *untouch*
- Intermediary connections remain constant throughout
- First AIG, then MIG (Recall: MIGs ⊃ AIGs)



Experimental Results

Classifier Accuracy -

- Overall prediction accuracy after training **79.27%**

79.78% - AIGs
(unbiased)
78.35% - MIGs

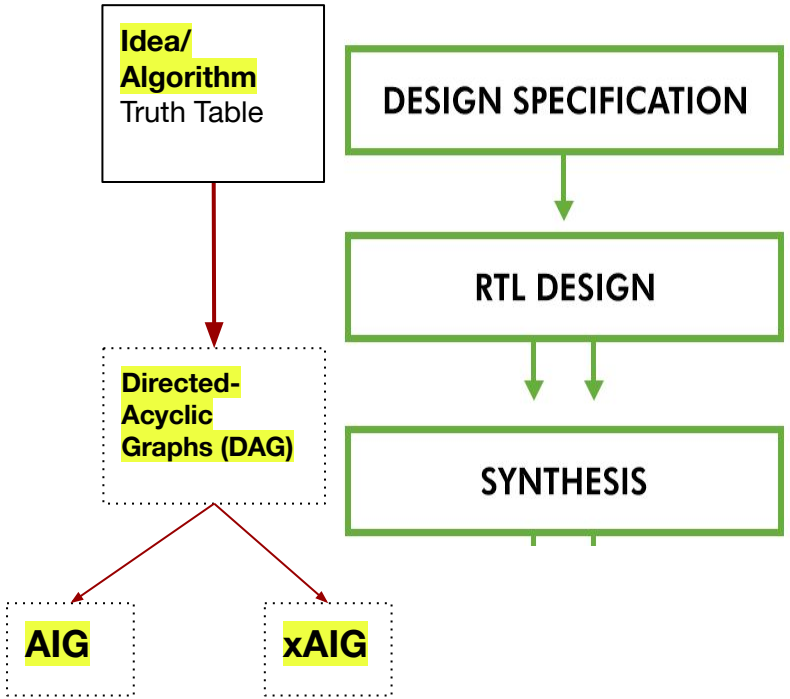
Performance Results -

Circuit	Technology-independent Results								ASIC Technology Mapped Results					
	Original		ABC		CirKit		LSOracle		ABC		CirKit		LSOracle	
	#nodes	depth	#nodes	depth	#nodes	depth	#nodes	depth	area	delay	area	delay	area	delay
Pico-RV	17,010	36	16,483	36	15,522	26	16,521	38	10,760	170	10,683	173	10,802	161
oc_aquarius	25,058	276	19,653	206	27,328	97	20,713	167	13,685	730.9	17,377	524.9	16,678	543.9
s38417	12,394	36	8,352	27	8,135	21	8,559	28	7,142	146.5	7,144	144	7,135	142.9
chip_bridge	124,565	29	72,190	26	64,876	22	70,456	29	42,094	198.9	43,784	189	41,512	187.9
FPU	64,814	145	61,424	145	58,445	44	64,469	131	47,573	206	48,318	197.9	47,124	200
sum:	243,841	522	178,102	440	174,306	210	179,718	393	121,256	1,452	127,307	1,228	123,252	1,235
ratio:	1.00	1.00	0.73	0.84	0.71	0.40	0.73	0.75	1.00	1.00	1.05	0.84	1.01	0.85

Learning to Design Efficient Logic Circuits

Adam Hillier, Ngan Vu

(Google DeepMind)



IWLS Contest 2023

Problem Statement:

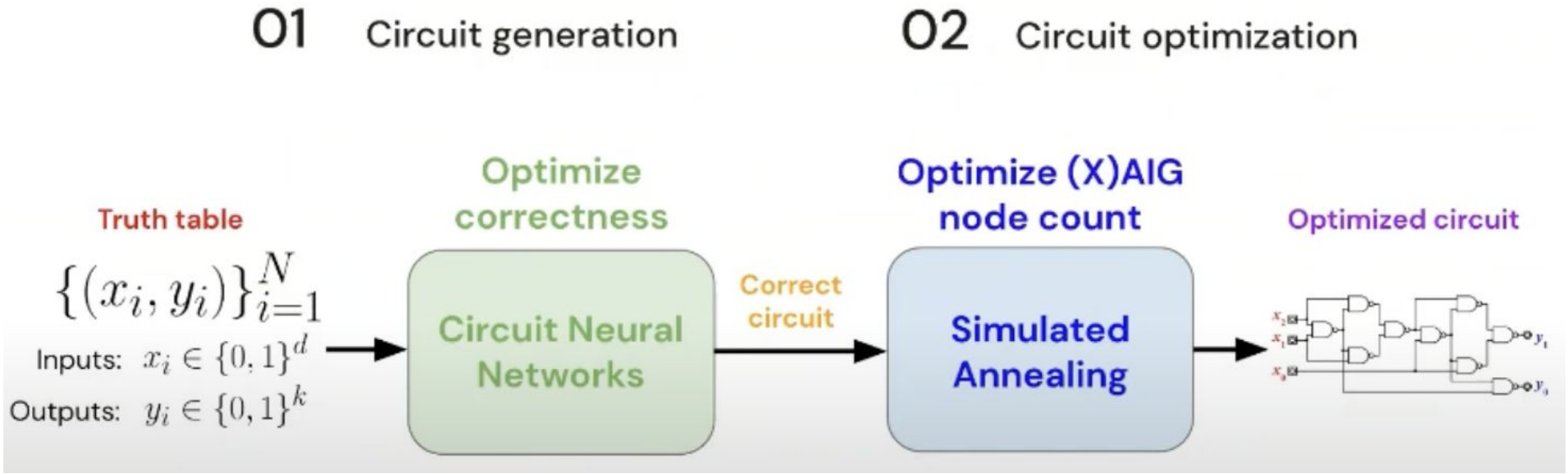
Given 100 boolean function truth tables, synthesize

- AIG (and-inverter graph)
- XAIG (xor-and-inverter graph)

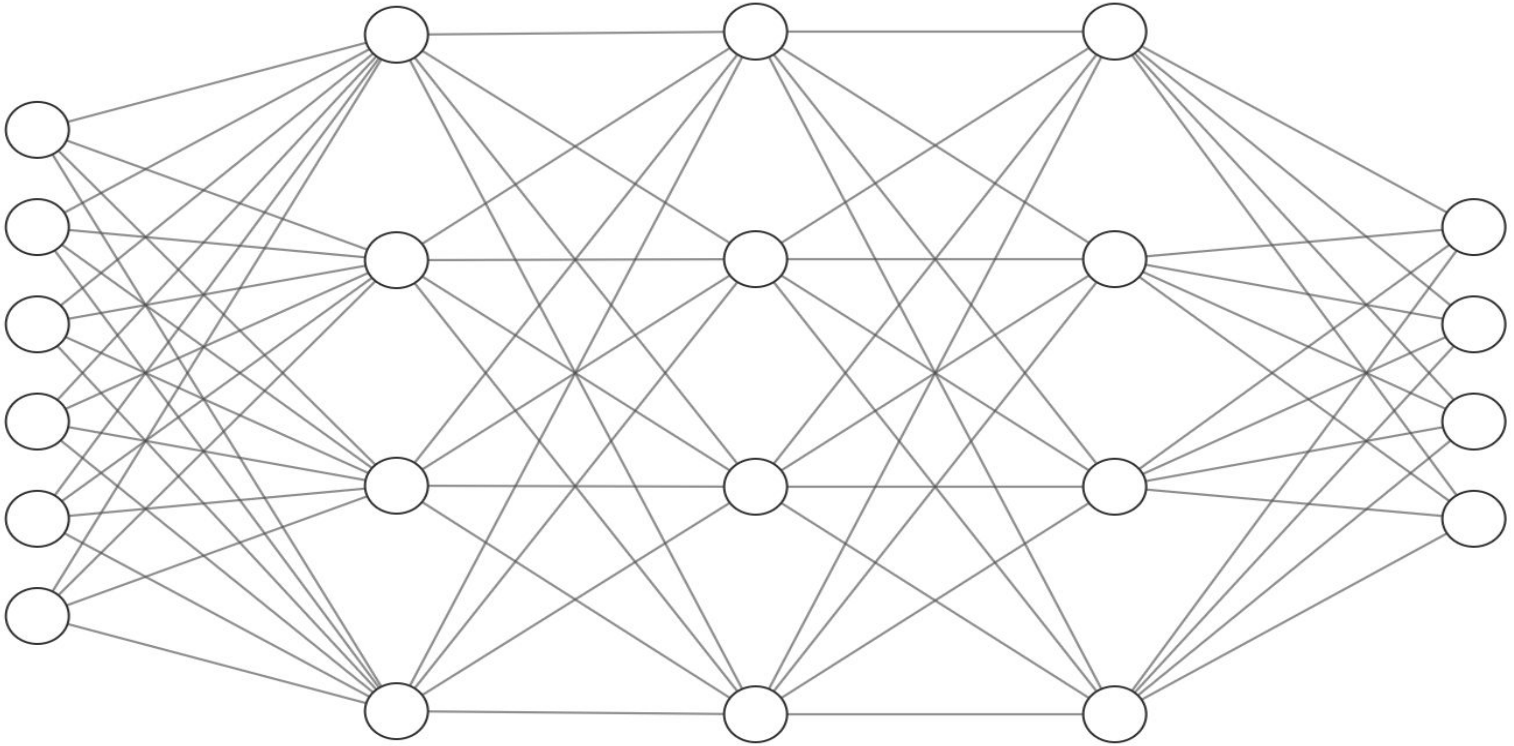
with the minimum number of nodes.

- Team EPFL:
  - Hanyu Wang, Alessandro Tempia Calvino, Siang-Yun Lee
- Google DeepMind:
  - Adam Hillier, Georges Rotival, Ivan Lobov, Kshiteej Mahajan, Marco Gelmi, Ngan Vu, Olivier Temam, Sergio Guadarrama, Vinod Nair
- TU Wien:
  - Franz-Xaver Reichl, Friedrich Slivovsky, Stefan Szeider
- NBU-Giga:
  - Chengyu Ma, Hongyang Pan, Ruibing Zhang, Yong Xiao, Yun Shao, Zhufei Chu

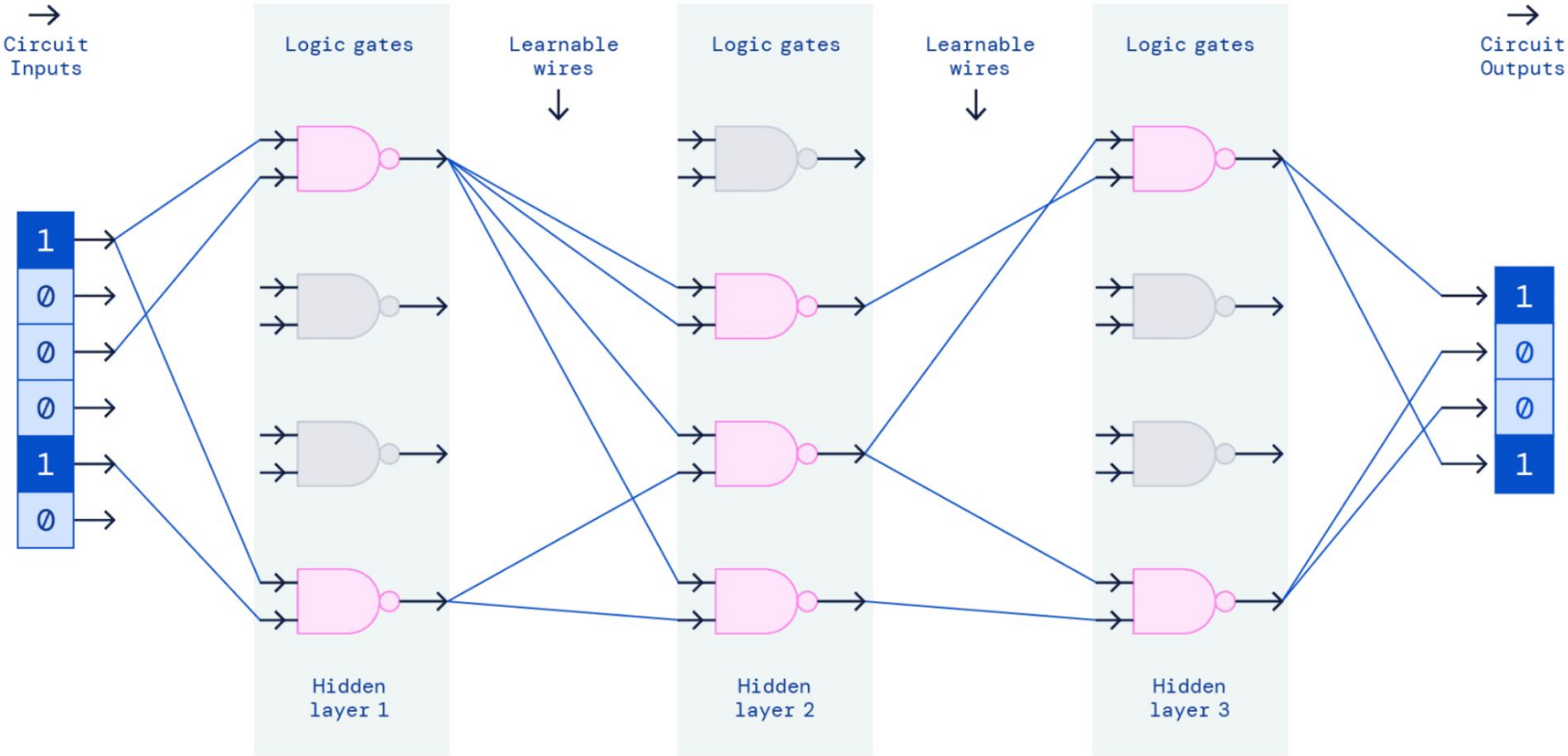
A Different CNN (Circuit Neural Networks)



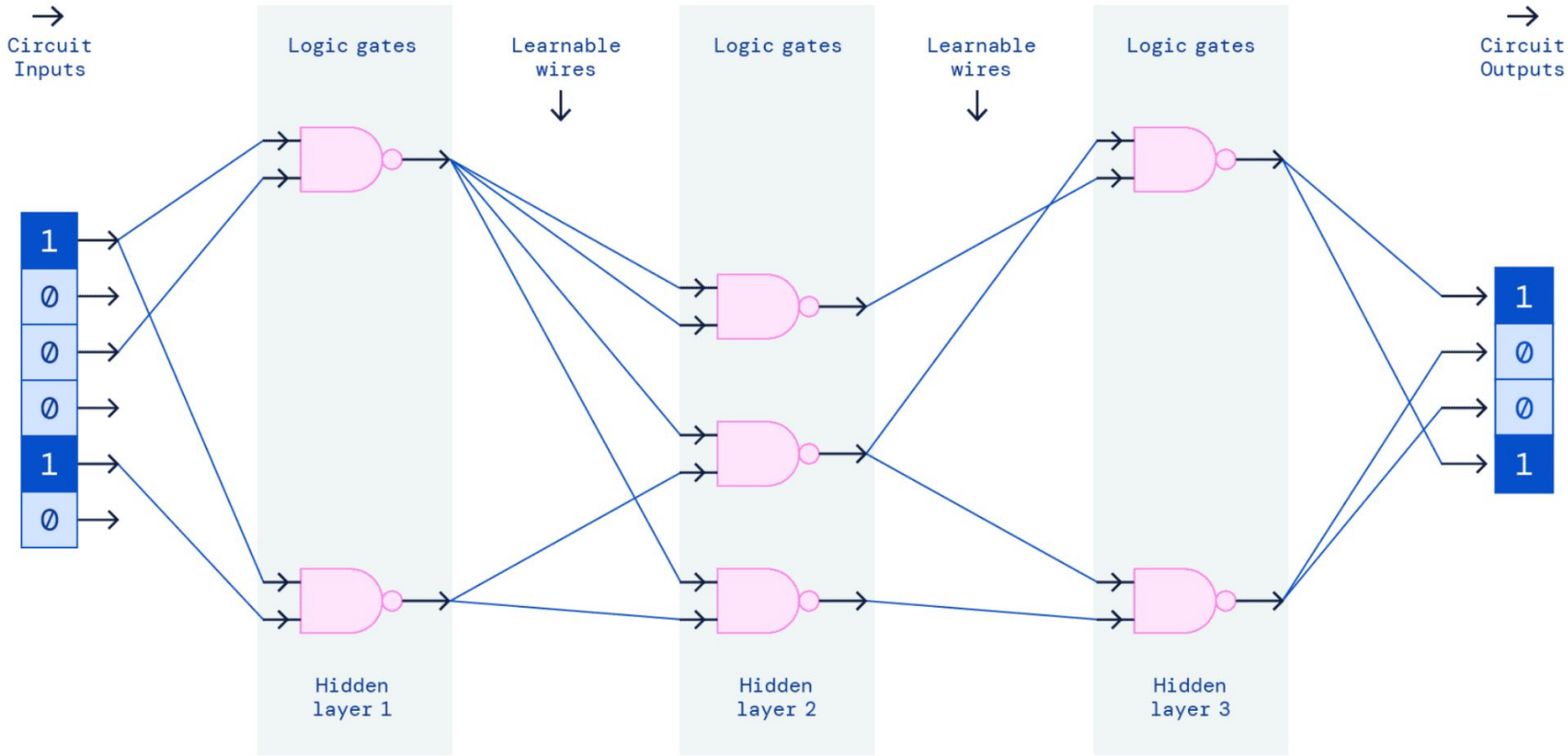
A Different CNN (Circuit Neural Networks)



A Different CNN (Circuit Neural Networks)



A Different CNN (Circuit Neural Networks)



Challenges and Solutions

- Logic gates are discrete functions (not differentiable)
  - Relax to differentiable logic gate (https://arxiv.org/pdf/2210.08277.pdf)

- How do we choose which edges to take?
  - Stochastic Process (now not differentiable!)
  - **Gumbel-Softmax Trick**

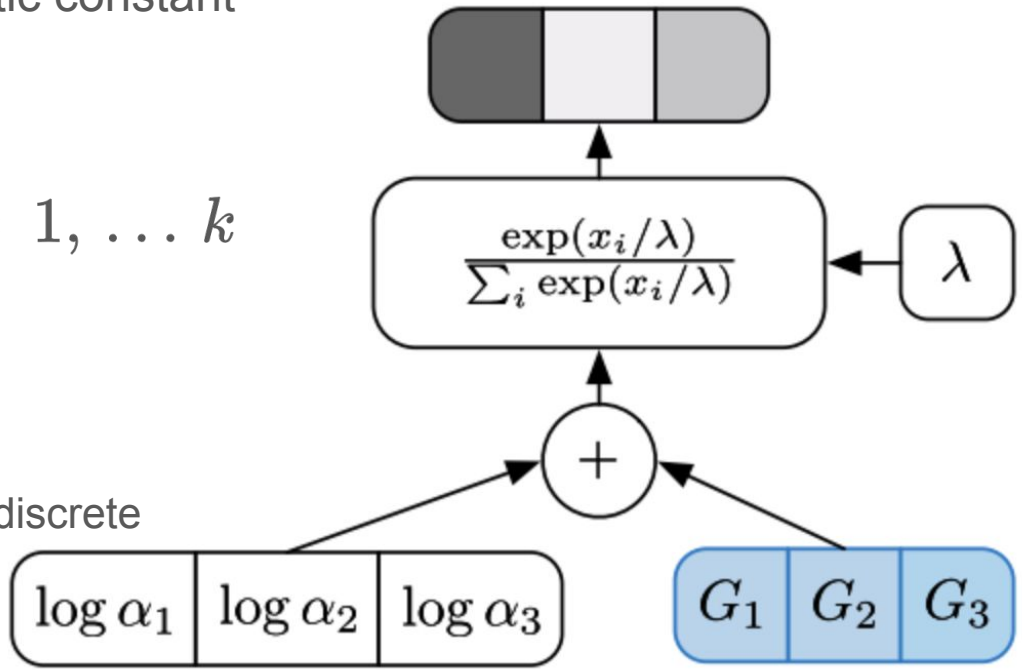
ID	Operator	real-valued	00	01	10	11
0	False	0	0	0	0	0
1	$A \wedge B$	$A \cdot B$	0	0	0	1
2	$\neg(A \Rightarrow B)$	$A - AB$	0	0	1	0
3	A	A	0	0	1	1
4	$\neg(A \Leftarrow B)$	$B - AB$	0	1	0	0
5	B	B	0	1	0	1
6	$A \oplus B$	$A + B - 2AB$	0	1	1	0
7	$A \vee B$	$A + B - AB$	0	1	1	1
8	$\neg(A \vee B)$	$1 - (A + B - AB)$	1	0	0	0
9	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$	1	0	0	1
10	$\neg B$	$1 - B$	1	0	1	0
11	$A \Leftarrow B$	$1 - B + AB$	1	0	1	1
12	$\neg A$	$1 - A$	1	1	0	0
13	$A \Rightarrow B$	$1 - A + AB$	1	1	0	1
14	$\neg(A \wedge B)$	$1 - AB$	1	1	1	0
15	True	1	1	1	1	1

Aside: Gumbel-Softmax Trick

Transform discrete probability distribution to differentiable approximation

"deterministic variable + stochastic constant"

$$y_i = \frac{e^{\frac{\log(\alpha_i)+g_i}{\lambda}}}{\sum_{j=1}^{k} e^{\frac{\log(\alpha_j)+g_j}{\lambda}}} \; for \; i = 1, \ldots k$$

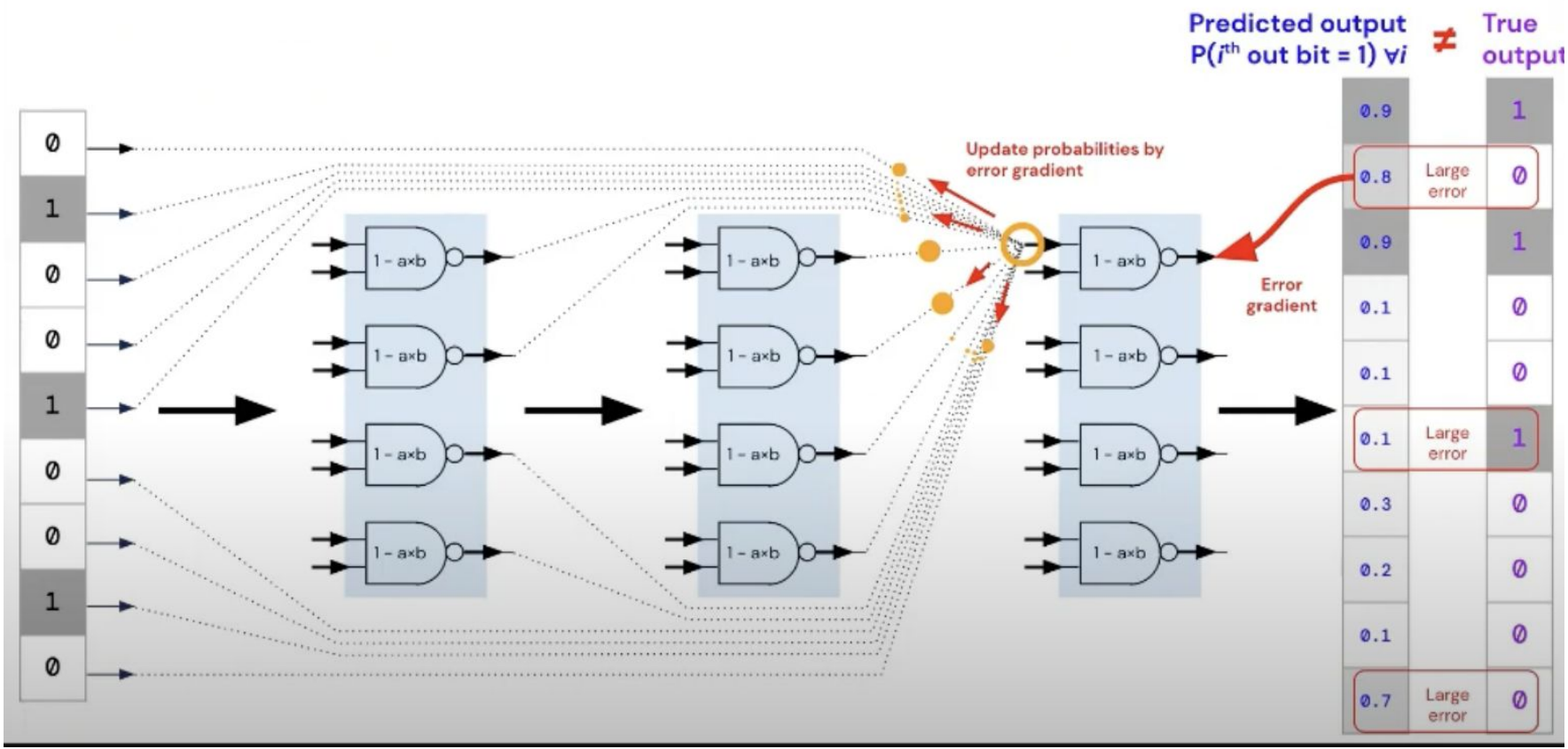


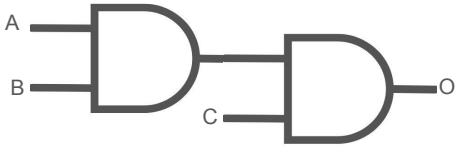
λ is a 'temperature factor'

As $\lambda \to 0$: Gumbel Softmax Approaches discrete

https://arxiv.org/pdf/1611.01144.pdf
https://sassafras13.github.io/GumbelSoftmax/

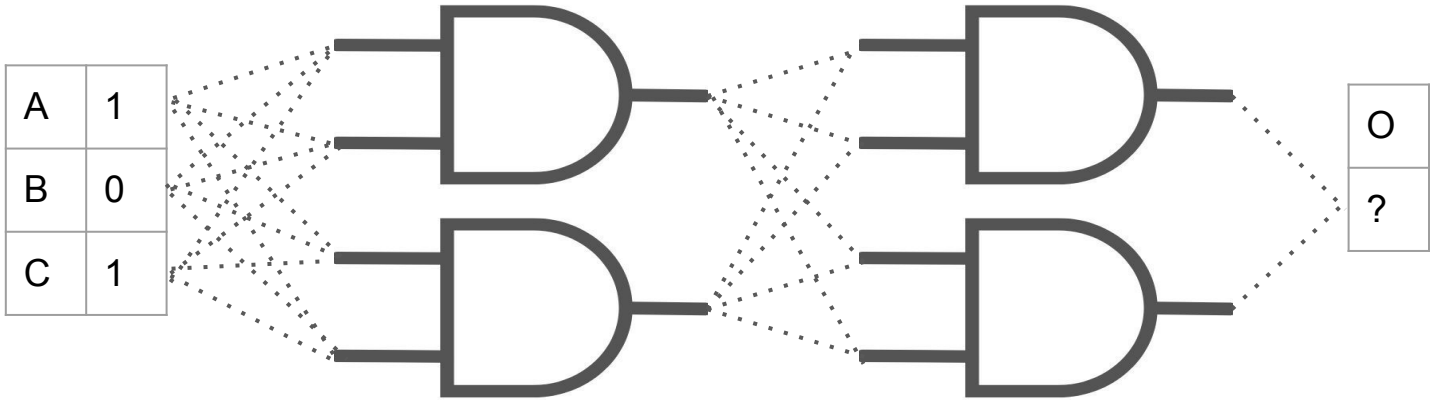
What Does Training Look Like?

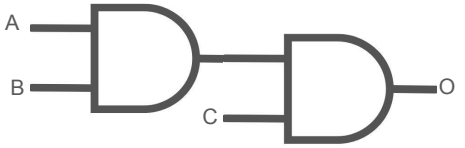




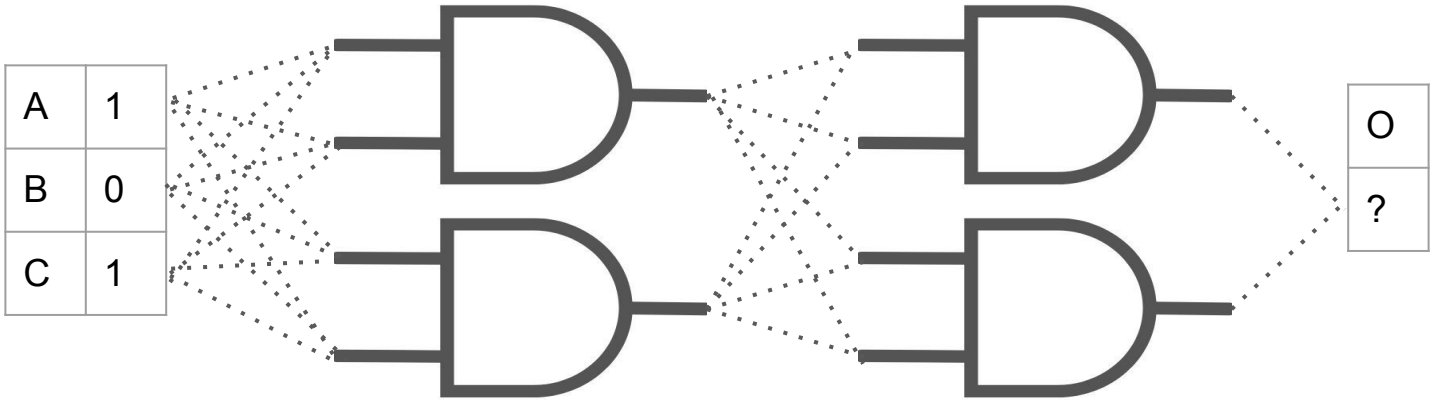
Example: AND3

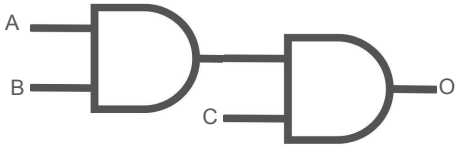
A	B	C	O
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



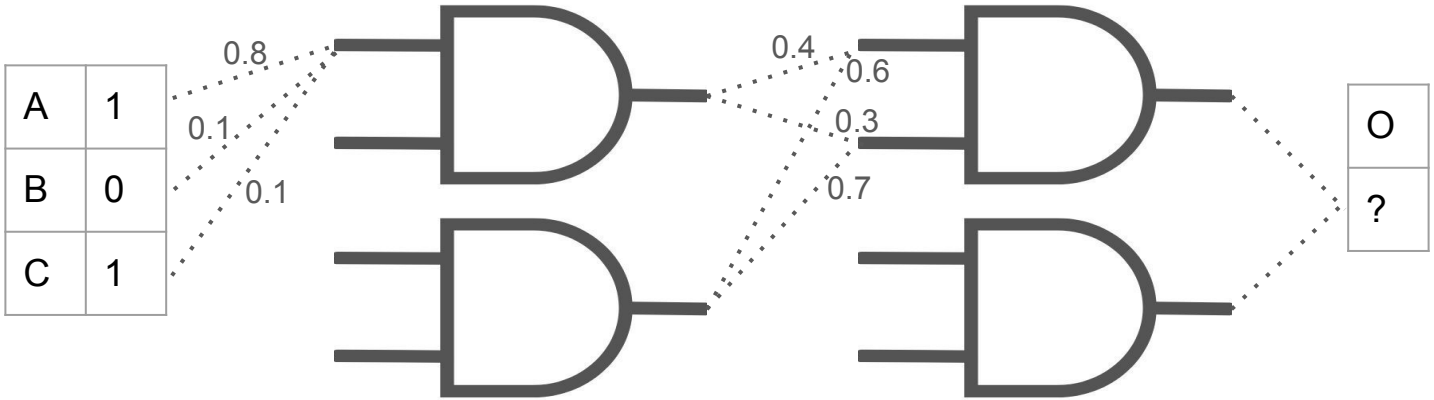


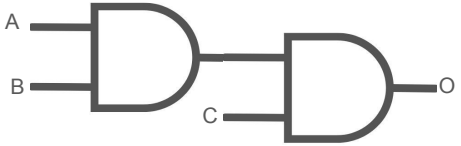
Example: AND3



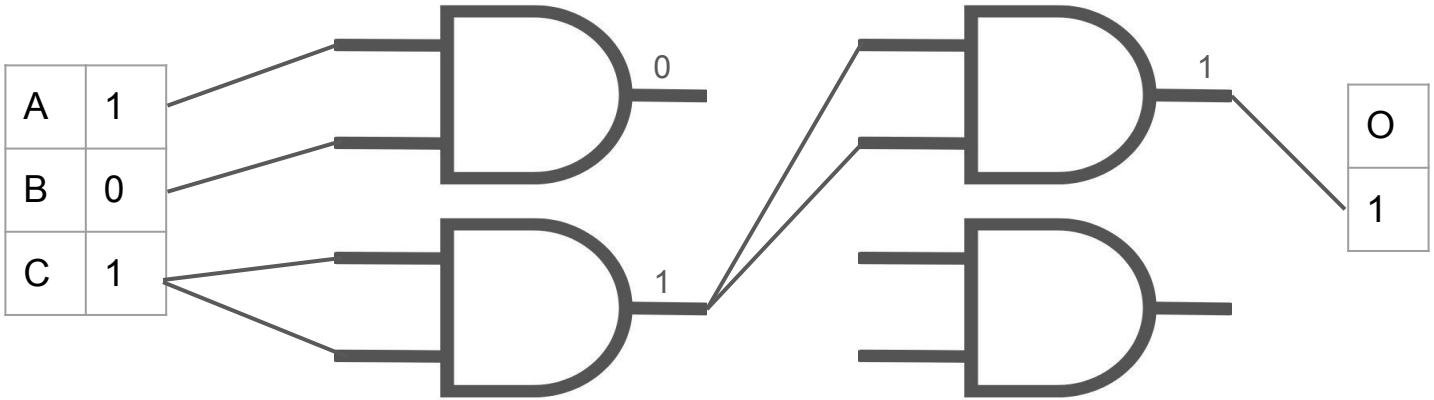


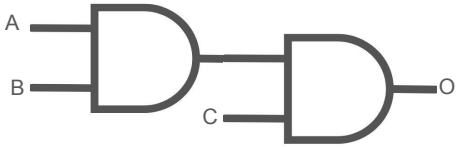
Example: Sample





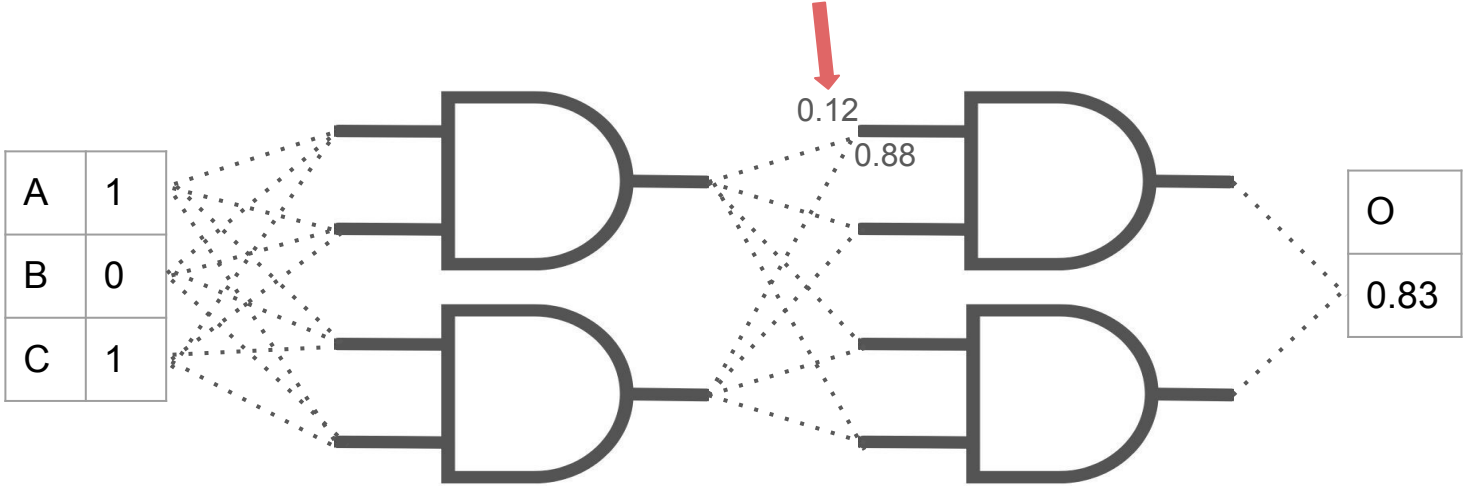
Example: Sample





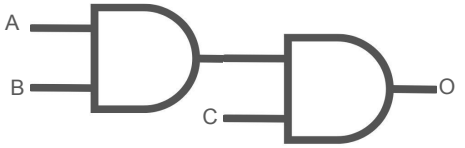
Example: Transform

$$y = \frac{\exp((\ln(0.6)+g)/\lambda)}{\exp((\ln(0.6)+g)/\lambda)+\exp((\ln(0.4)+g)/\lambda)},\ \lambda = 0.2$$

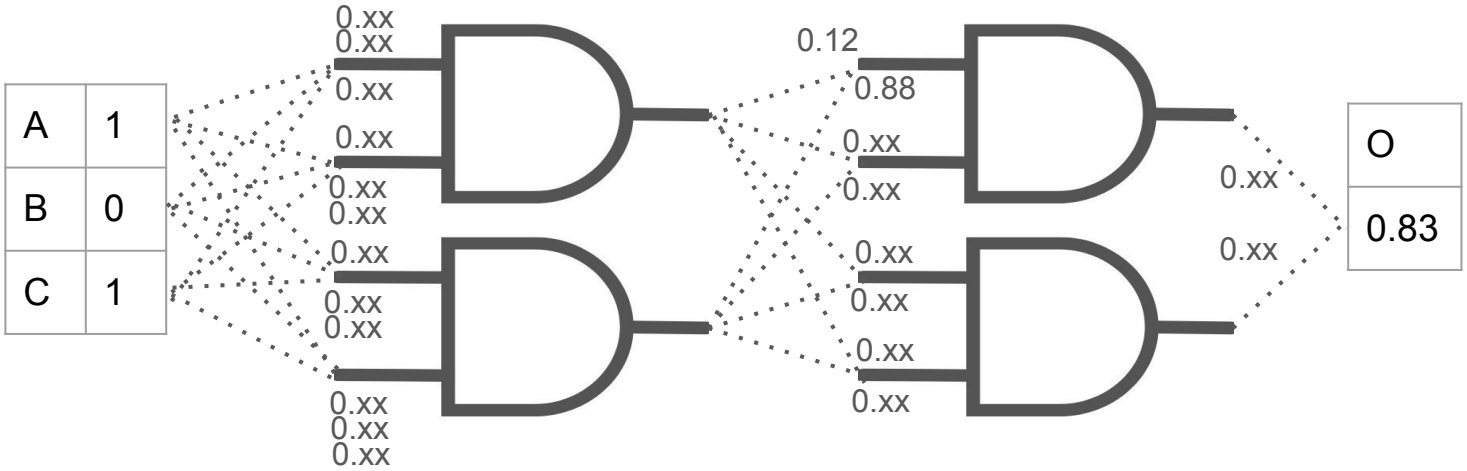


A	1
B	0
C	1

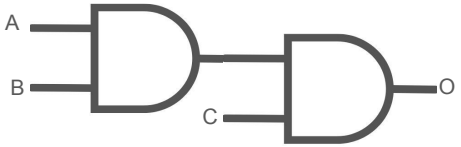
O
0.83

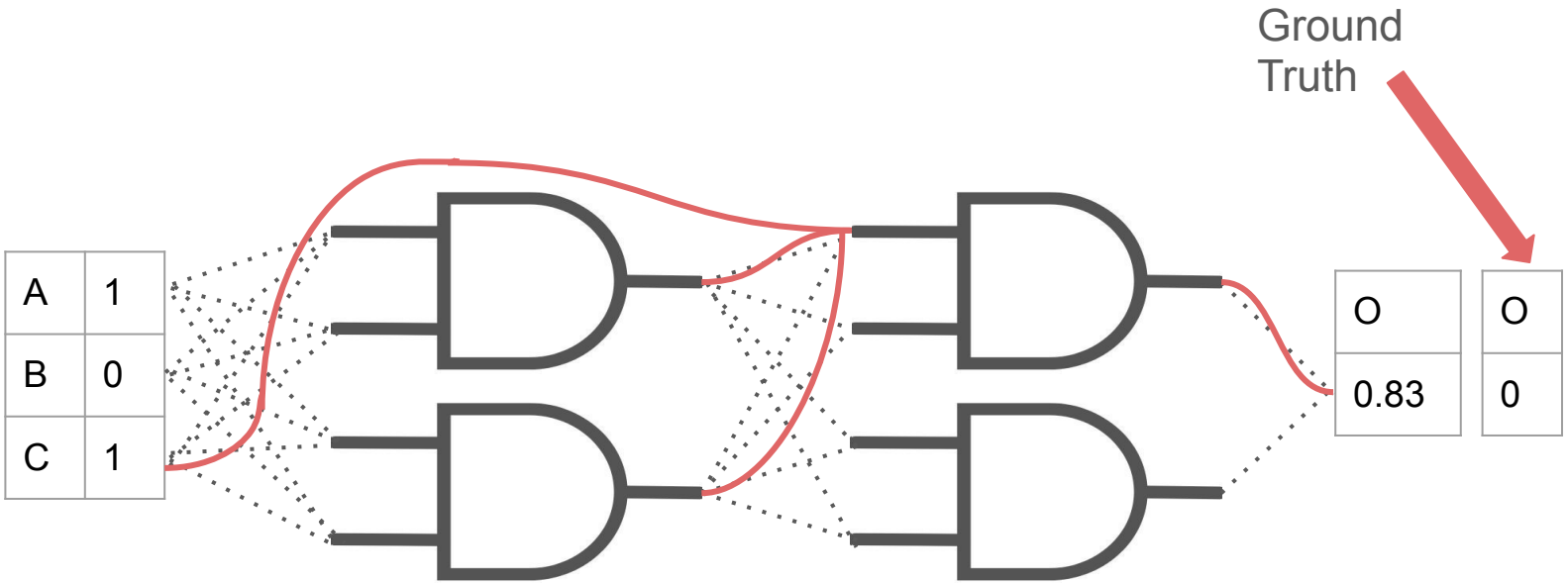


Example: Transform

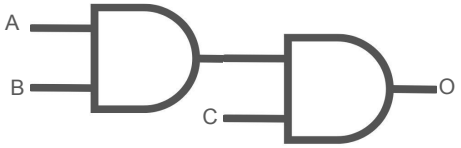


Example: Back Propagation

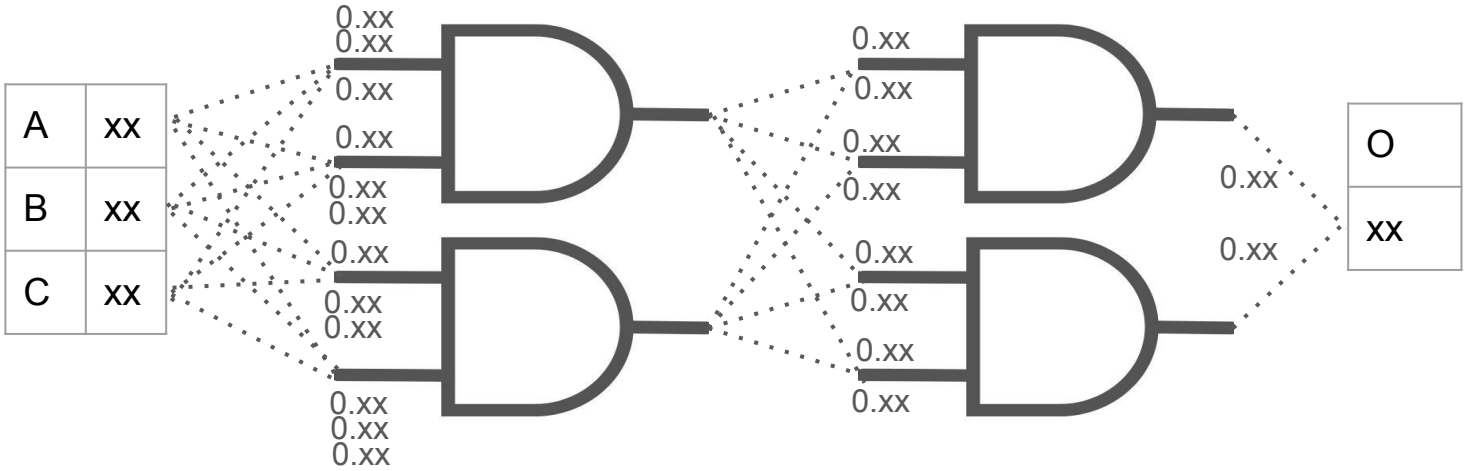


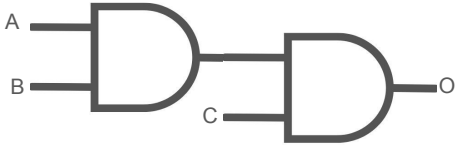



A FEW
MOMENTS LATER

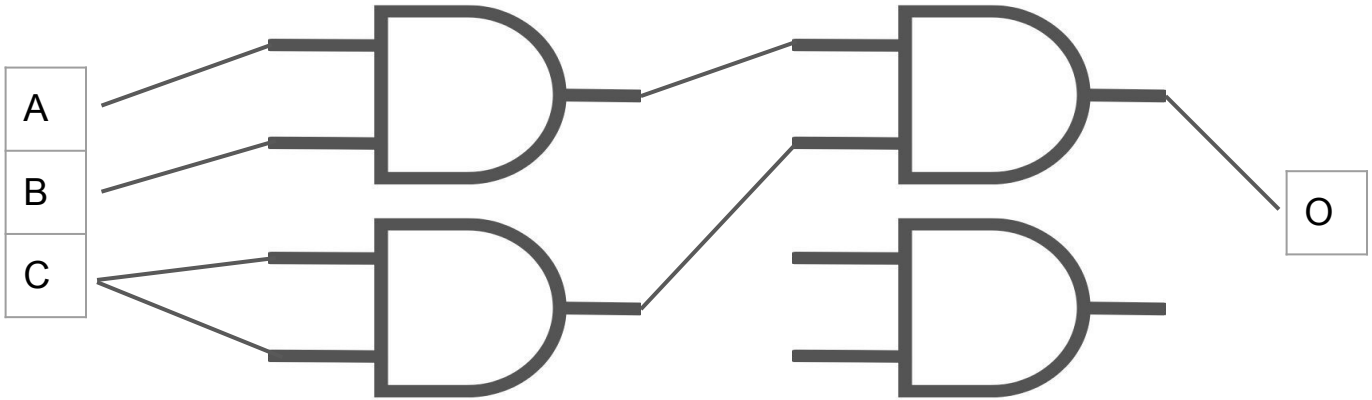


Example: Final Weights

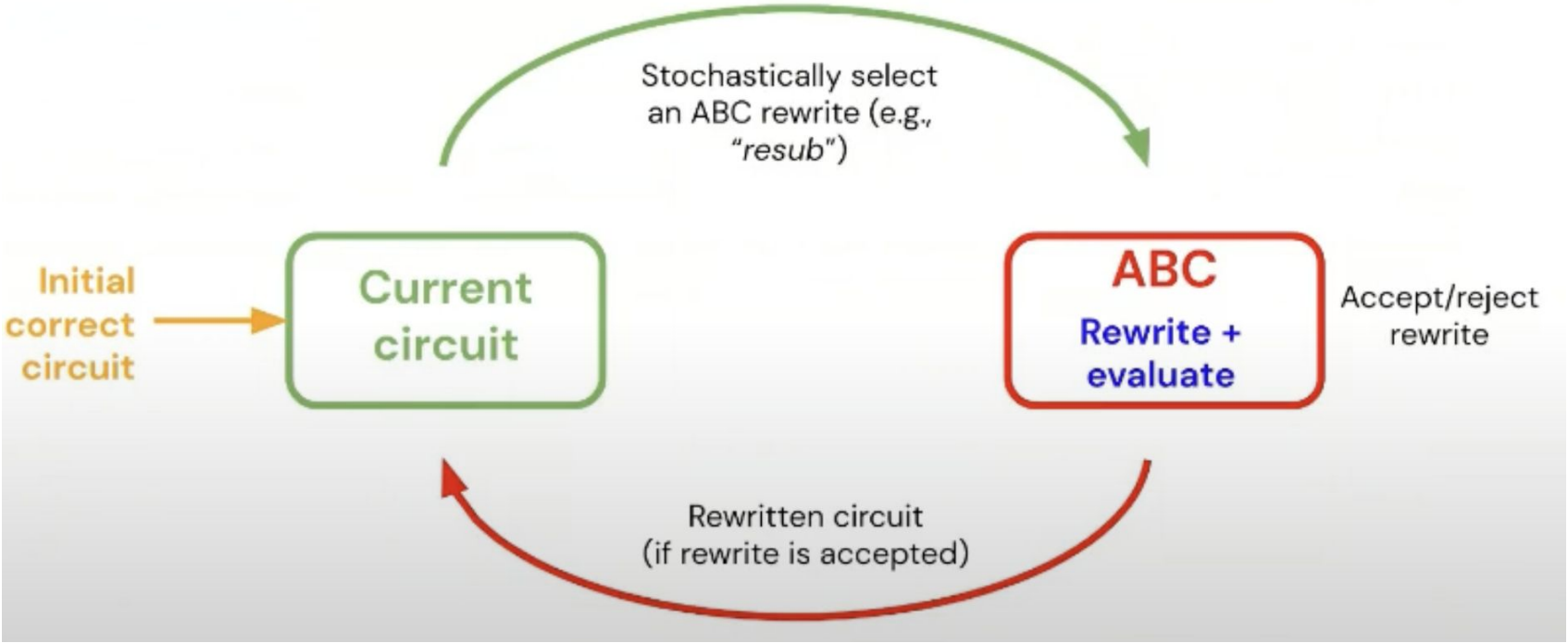




Example: Final Sample



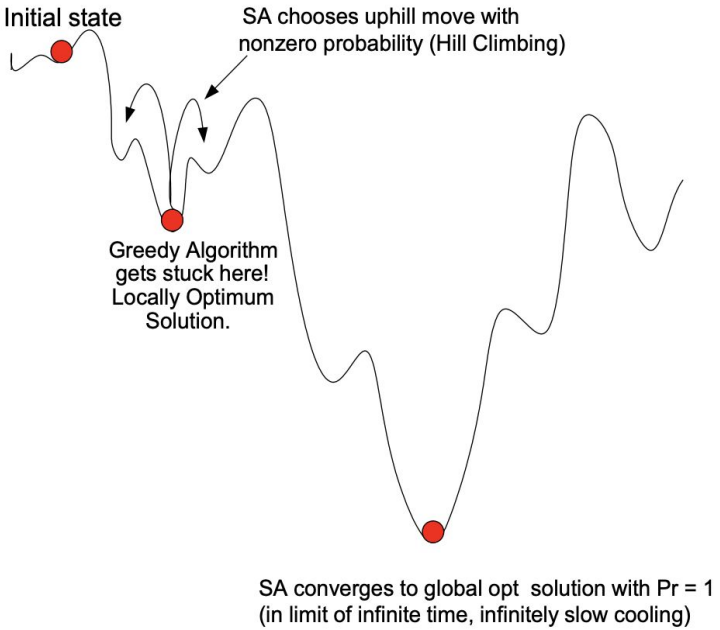
Step 2: Optimize with Simulated Annealing



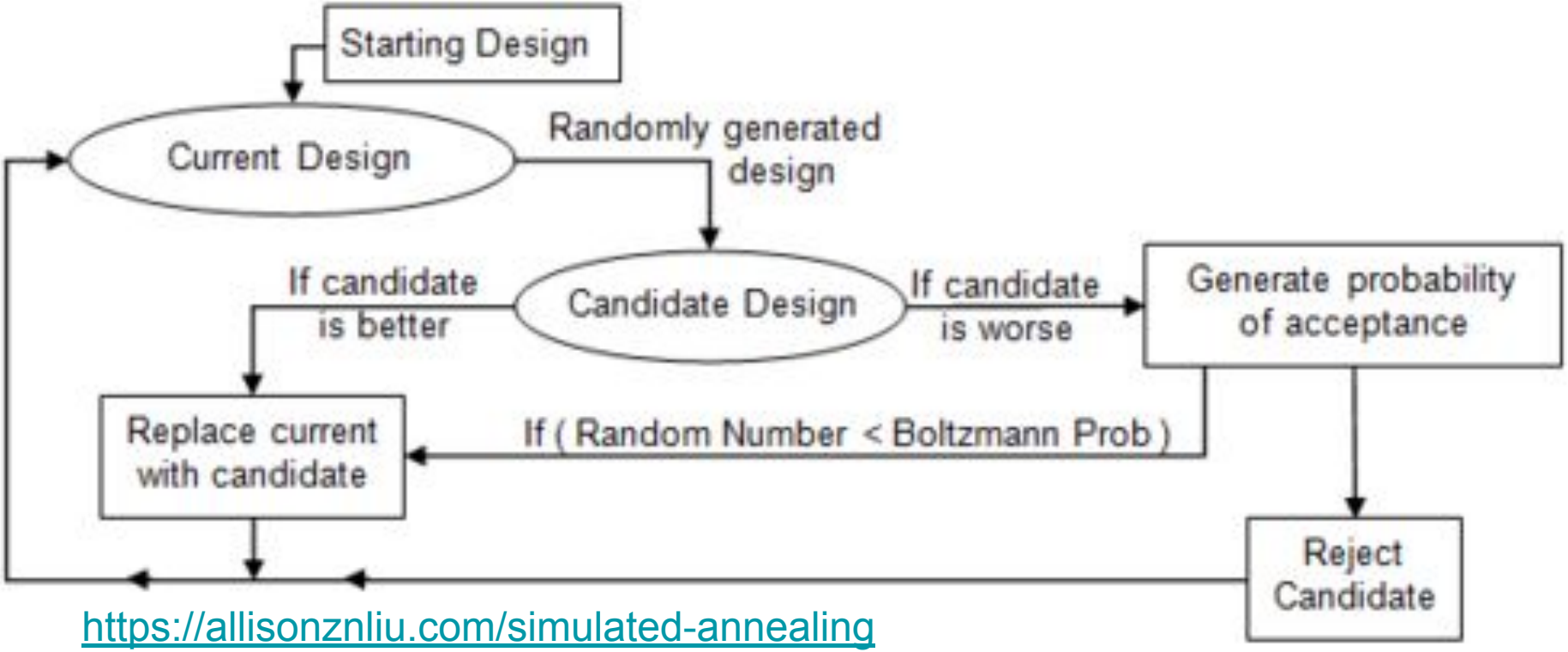
Step 2: Optimize with Simulated Annealing

1. Start with an arbitrary solution S
2. Make a small change to the solution $S' = S + \epsilon$
3. If S' is better than S:
   a. Accept S' as new solution S
4. Else
   a. Accept S' as new solution S with probability T
5. Repeat until end condition

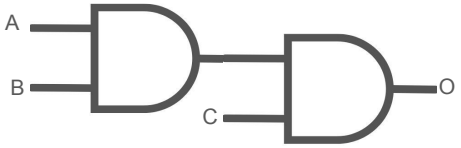
*Lower T over time



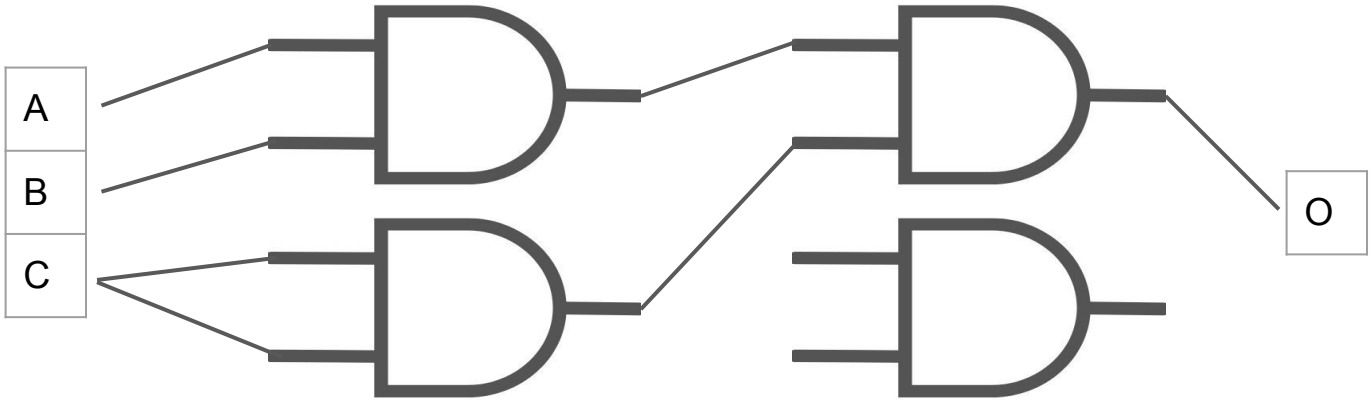
Simulated Annealing: Bonus Slide



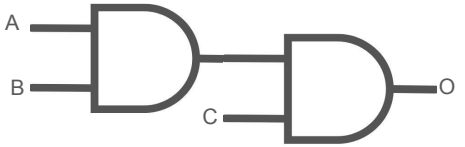
https://allisonznliu.com/simulated-annealing
https://apmonitor.com/me575/index.php/Main/SimulatedAnnealing



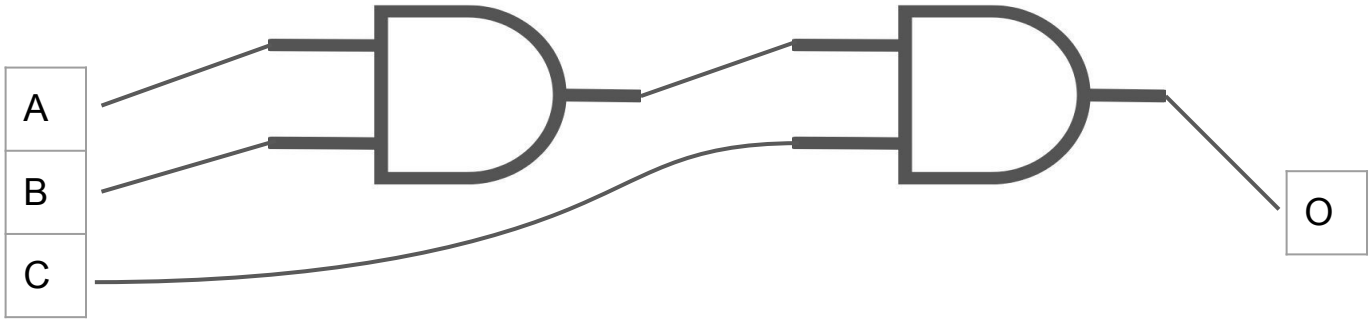
Example: “Arbitary Solution”



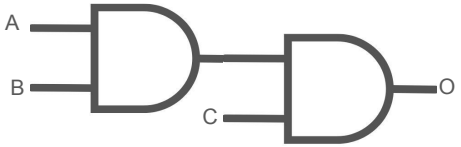
*Note: I am skipping the complexities of simulated annealing for the sake of time



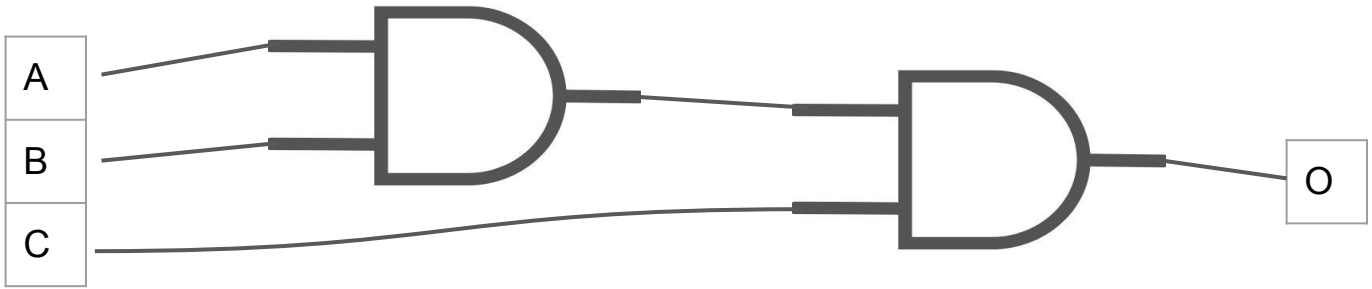
Example: “Small Change”



*Note: I am skipping the complexities of simulated annealing for the sake of time

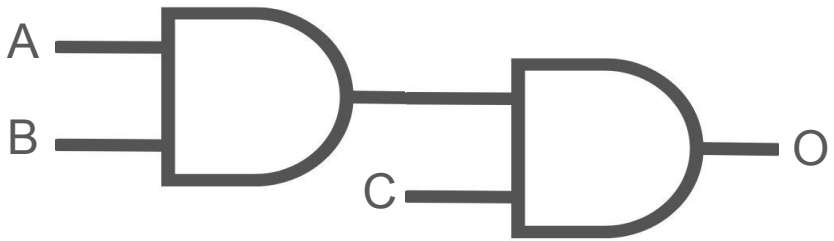


Example: Better So Keep



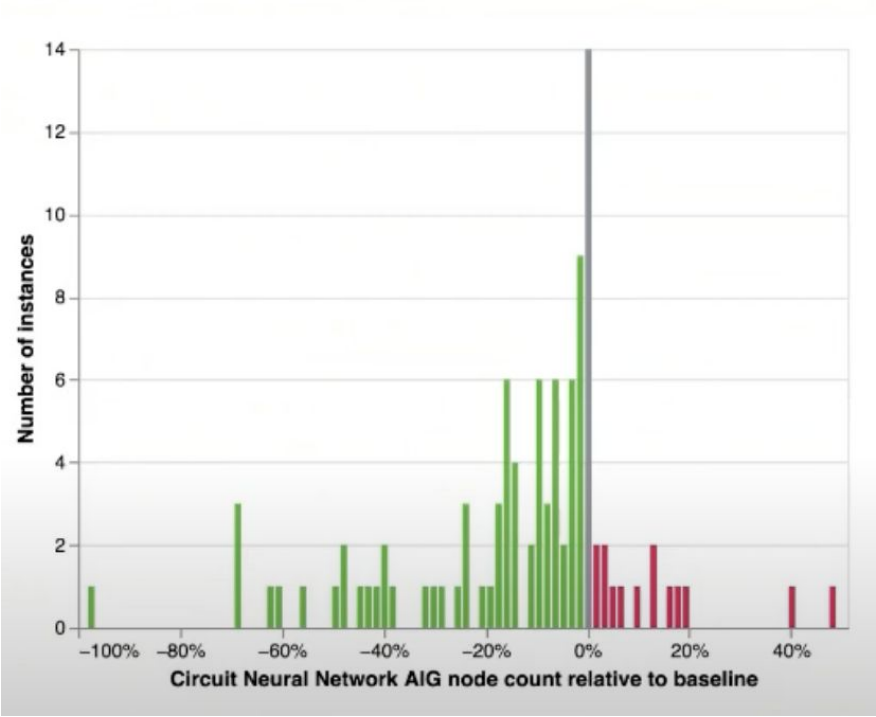
*Note: I am skipping the complexities of simulated annealing for the sake of time

Example: Final Solution

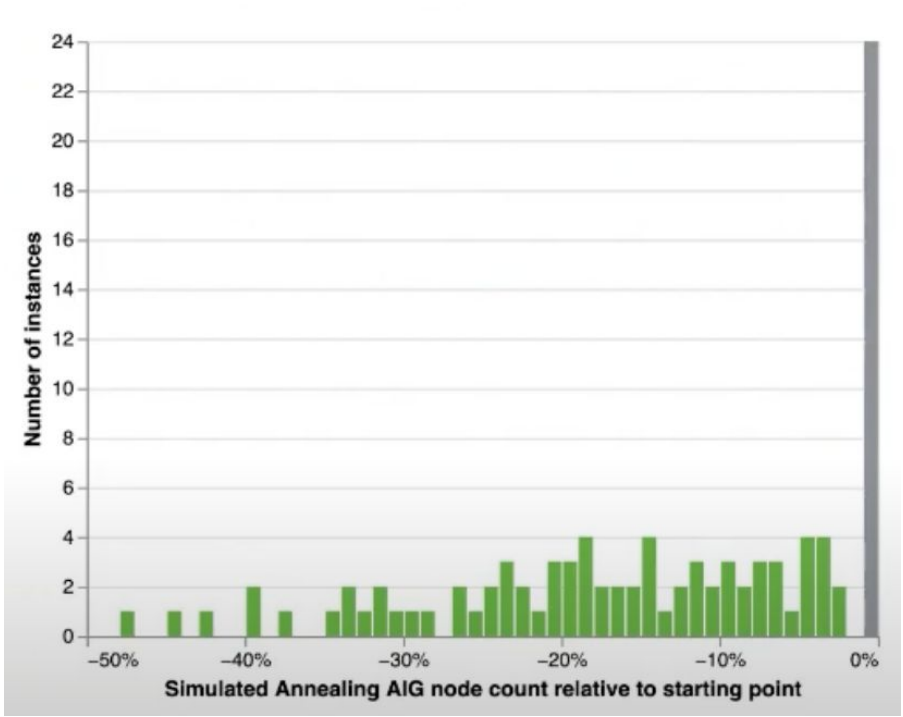


Results

1. Circuit Neural Net Generation



2. Simulated Annealing

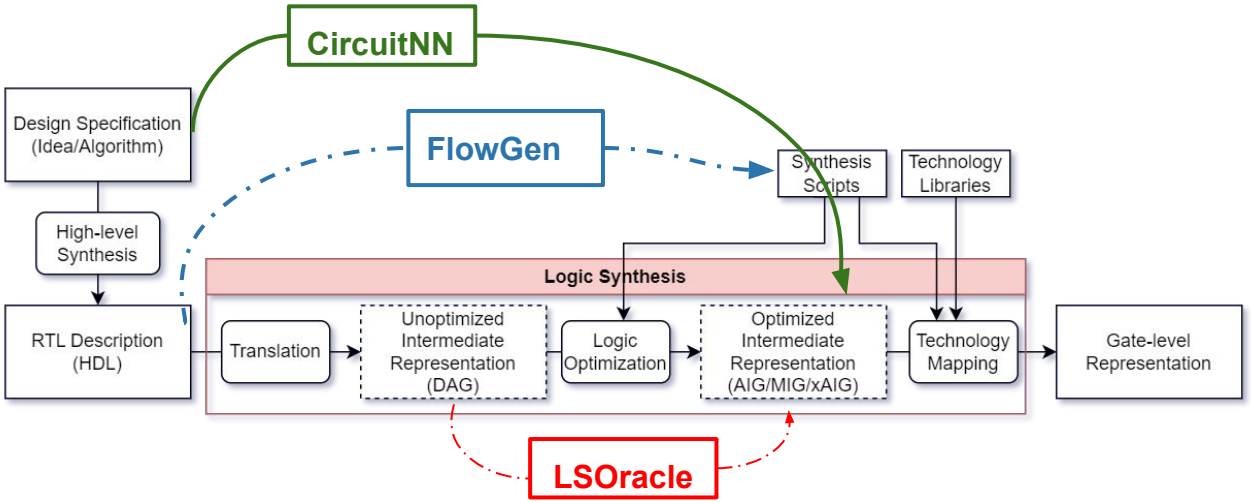


Limitations

- Slow to generate Circuits.
  - Especially for larger circuits
  - Number/type of circuits in layers is a hyperparameter that should be swept

- Does not generate ideal Circuit, hence Simulated Annealing required
  - Circuit NN is not an end-all solution, but it does generate a strong starting point!

Summary



FlowGen

- Find which synthesis scripts have best/worst QoR
- Generates random synthesis flows and their QoR
- CNN to classify flows as angel and devil

LSOracle

- Optimize the synthesized logic network
- Partition DAG then create Karnaugh Map image
- MLP classification into AIG or MIG optimizations
- Recombine partitioned DAGs into optimized DAG

CircuitNN

- Create an optimized logic network from a truth table
- Build AIG/XAIG from probabilistic ML Model
- Use Simulated Annealing to further optimize candidate circuits

References

C. Yu, H. Xiao, and G. De Micheli, “Developing synthesis flows without human knowledge,” in Proceedings of the 55th Annual Design Automation Conference, 2018, pp. 1–6.

W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, “LSOracle: A logic synthesis framework driven by artificial intelligence,” in ICCAD. IEEE, 2019, pp. 1–6

DeepMind Competition Results and Presentation at IWLS23 (Int’l Workshop for Logic Synthesis) : https://www.youtube.com/watch?v=ouyyM2hQxbo

Questions?