Circuit Transformer: A Transformer That Preserves Logical Equivalence

Xihan Li[1], Xing Li[2], Lei Chen[2], Xing Zhang[2], Mingxuan Yuan[2] and Jun Wang[1]

1. Department of Computer Science, University College London
2. Huawei Noah's Ark Lab, Hong Kong, China

Speaker: Xihan Li (xihan.li@cs.ucl.ac.uk)

The 13th International Conference on Learning Representations (ICLR 2025)

ICLR
International Conference On Learning Representations

Problem Setting

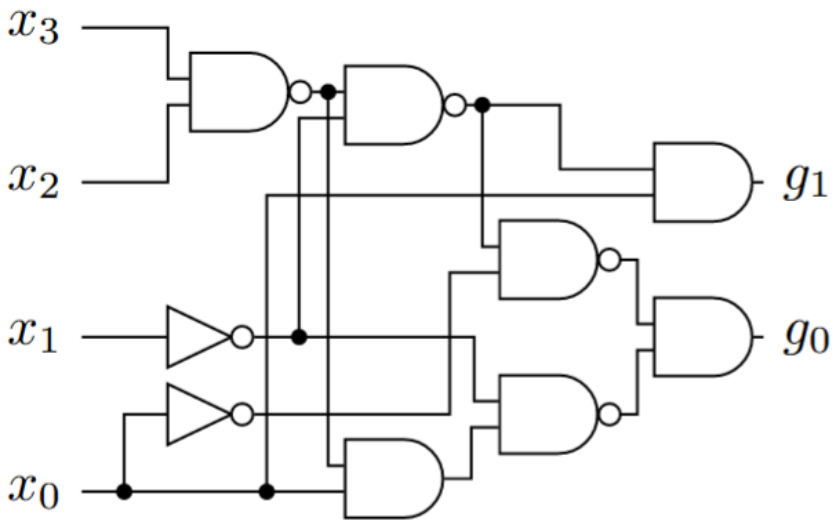
Implementing Boolean functions with logic circuits

(A fundamental problem in digital design!)

x_3	x_2	x_1	x_0	y_1	y_0	x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	0	0
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	0	1	1	1	0	0	0	0
0	1	0	1	0	0	1	1	0	1	1	1
0	1	1	0	0	0	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1

A Boolean function $(y_1, y_0) = f(x_3, x_2, x_1, x_0)$
in which $x_0, x_1, x_2, x_3, y_0, y_1 \in \{0,1\}$

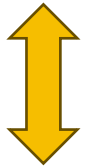
(represented by a truth table, which lists the value of (y_1, y_0) for all 16 possible inputs)



A logic circuit with 7 AND (NAND) gates that exactly implements f

The Challenge: Preserving Logical Equivalence

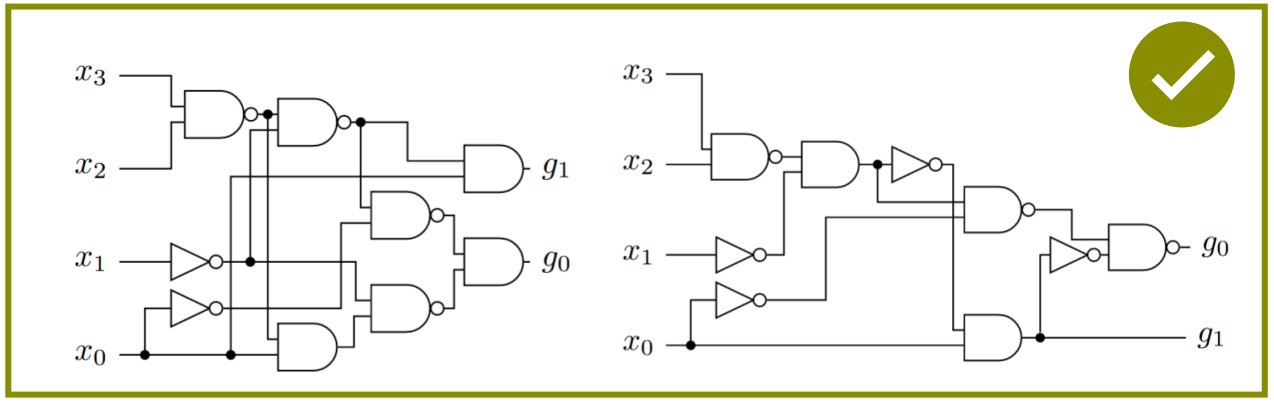
Boolean function f



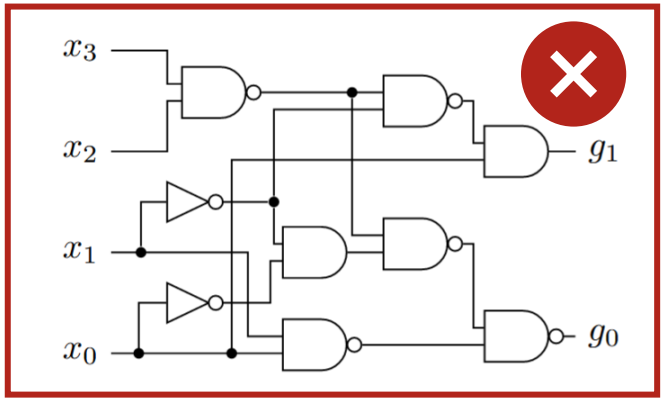
Must be **exactly** equivalent (i.e., without a single bit of error!)

x_3	x_2	x_1	x_0	y_1	y_0	x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	0	0
0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	0	0	1	1	1	0	0	0	0
0	1	0	1	0	0	1	1	0	1	1	1
0	1	1	0	0	0	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1

Logic circuit g



Outputs match for all 16 possible inputs



Outputs not exactly match

The Challenge: Preserving Logical Equivalence

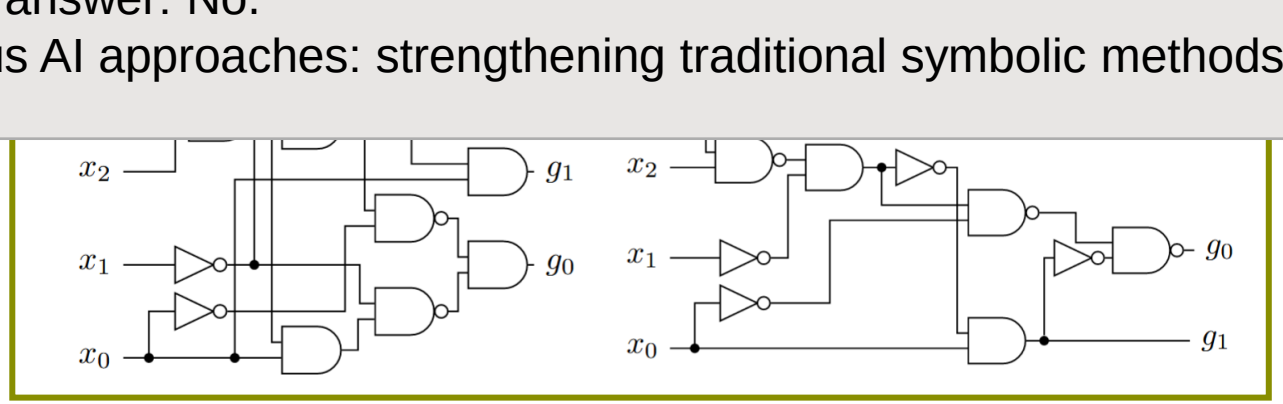
Boolean
funct

x_3	x_2	x_1	x_0	y_1	y_0	x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	0	0	1	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	0	1	0	0	0

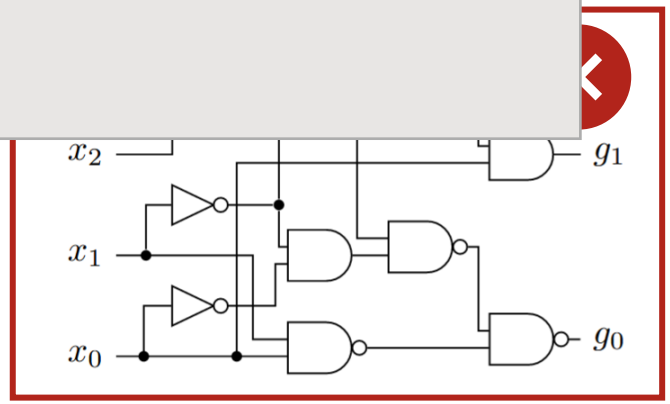
Lo
circu

Question: Can generative neural models directly generate strictly equivalent circuits?

Typical answer: No.
Previous AI approaches: strengthening traditional symbolic methods.



Outputs match for all 16 possible inputs



Outputs not exactly match

Motivation: Stepwise decomposition of a property

Typical Gen AI models (e.g., Transformer) are step-by-step sequence generators.

If we want to generate a sequence $s_1, s_2, \dots, s_N$ with certain property $F(s_1, \dots, s_N)$ strictly holds, we can decompose F as N "cutoff properties":

- $F(s_1)$
- $F(s_1, s_2)$
- $F(s_1, s_2, s_3)$
- ...
- $F(s_1, s_2, \dots, s_N)$

hold → hold → hold → hold

Step-by-step "partial feasible" construction towards a full feasible solution

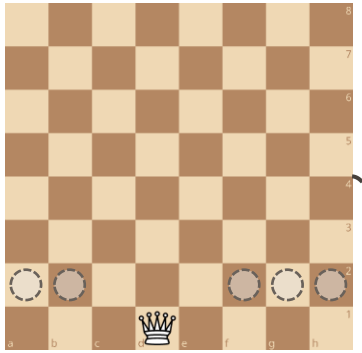
In this work, similar to the 8-queen problem, we incrementally construct a logic circuit with "cutoff properties" that preserve equivalence

Example: the 8-queen problem

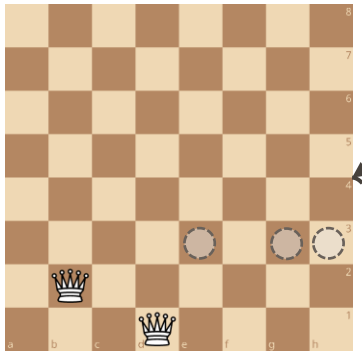
Desired property: all the 8 queens do not attack each other

$F(a, b, c, \dots)$: queens at a, b, c, … do not attack each other

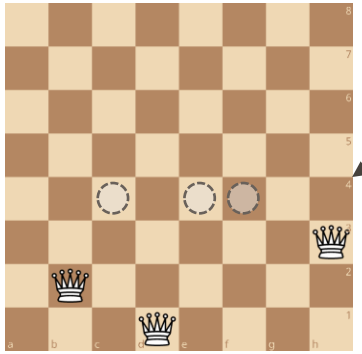
Step 1: $F(d1)$ holds
Queen at d1 will not attack each other
Easy to check
$F(d1, a2), F(d1, b2), F(d1, f2), F(d1, g2), F(d1, h2)$ hold
(valid choices $S_1 = \{a2, b2, f2, g2, h2\}$)
Choose $b2 \in S_1$



Step 2: $F(d1, b2)$ holds
Queens at d1, b2 will not attack each other
Easy to check
$F(d1, b2, e3), F(d1, b2, g3), F(d1, b2, h3)$ hold
(valid choices $S_2 = \{e3, g3, h3\}$)
Choose $h3 \in S_2$

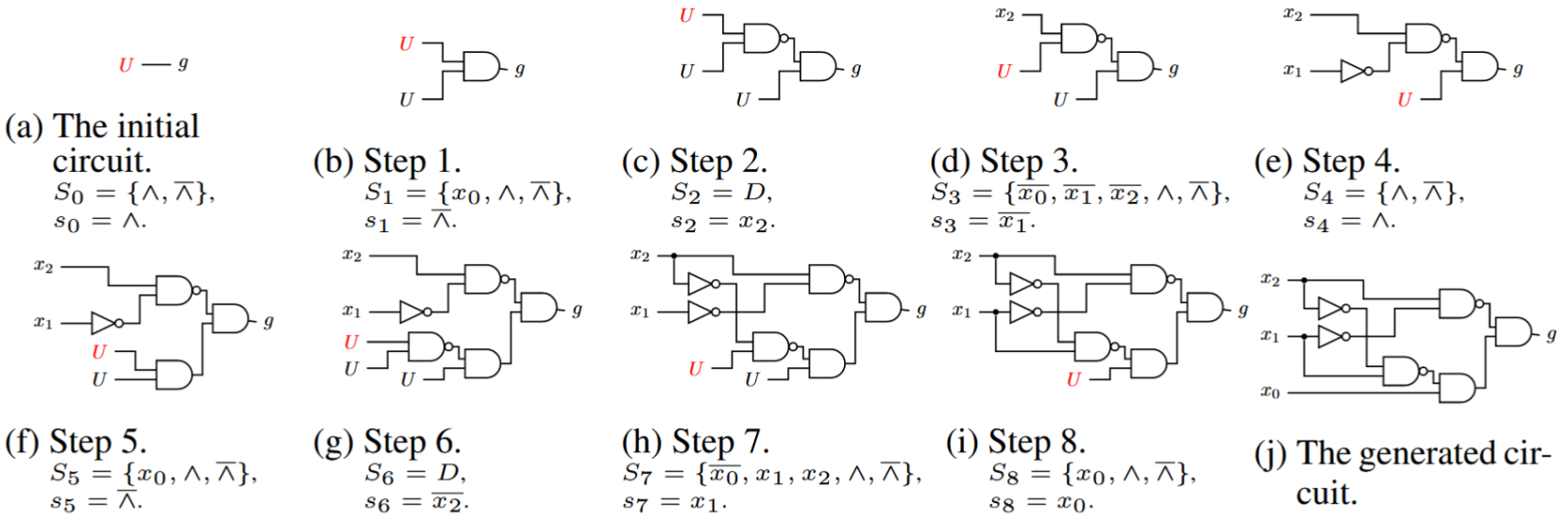


Step 3: $F(d1, b2, h3)$ holds
Queens at d1, b2, h3 will not attack each other
Easy to check
$F(d1, b2, h3, c4), F(d1, b2, h3, e4), F(d1, b2, h3, f4)$ hold
(valid choices $S_3 = \{c4, e4, f4\}$)
…



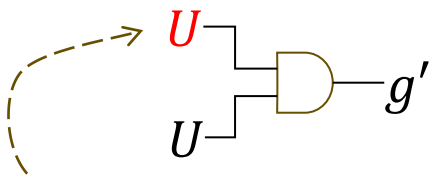
A Sequential Representation of Circuits with “Cutoff Properties”

- Construct from outputs to inputs to allow equivalence validation throughout the construction process.
- Initialize a circuit by a wildcard node (U)
- In each step, refine the circuit by replacing a wildcard node with a new gate / input
  - All the inputs of a new gate will be initialized to wildcard nodes
- When multiple wildcard node exist, follow a fixed order to replace them
  - Prioritizes those with the largest distance from the output
  - Prioritizes the left child of a gate over the right one



Computation of Valid Moves S_t

- Attempt to replace U by each possible input / gate
- Compute the output of the current circuit
- If no equivalence conflict occurs, then add it to S_t
  - U (unknown) will not conflict with any output
- This process can be done efficiently with cache and matrix operation



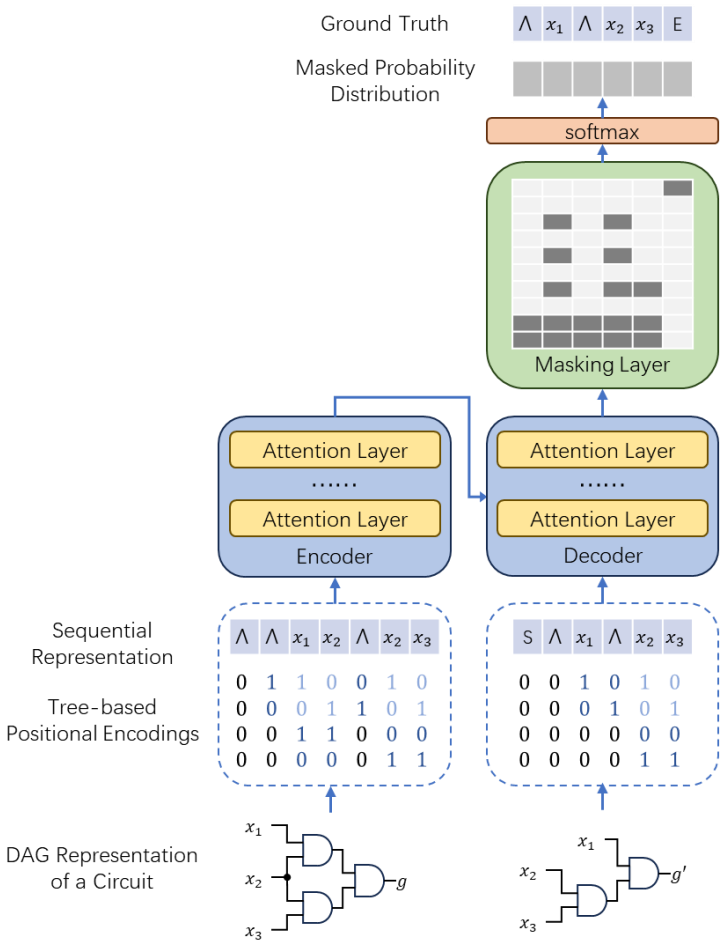
Replace U with each possible input / gate to see whether any equivalence conflict occurs

x_2	x_1	x_0	f	The value of g' when U is replaced by							
				x_0	$\overline{x_0}$	x_1	$\overline{x_1}$	x_2	$\overline{x_2}$	$\wedge$	$\overline{\wedge}$
0	0	0	**0**	0	U	0	U	0	U	U	U
0	0	1	**1**	U	0	0	U	0	U	U	U
0	1	0	**0**	0	U	U	0	0	U	U	U
0	1	1	**0**	U	0	U	0	0	U	U	U
1	0	0	**0**	0	U	0	U	U	0	U	U
1	0	1	**0**	U	0	0	U	U	0	U	U
1	1	0	**0**	0	U	U	0	U	0	U	U
1	1	1	**1**	U	0	U	0	U	0	U	U

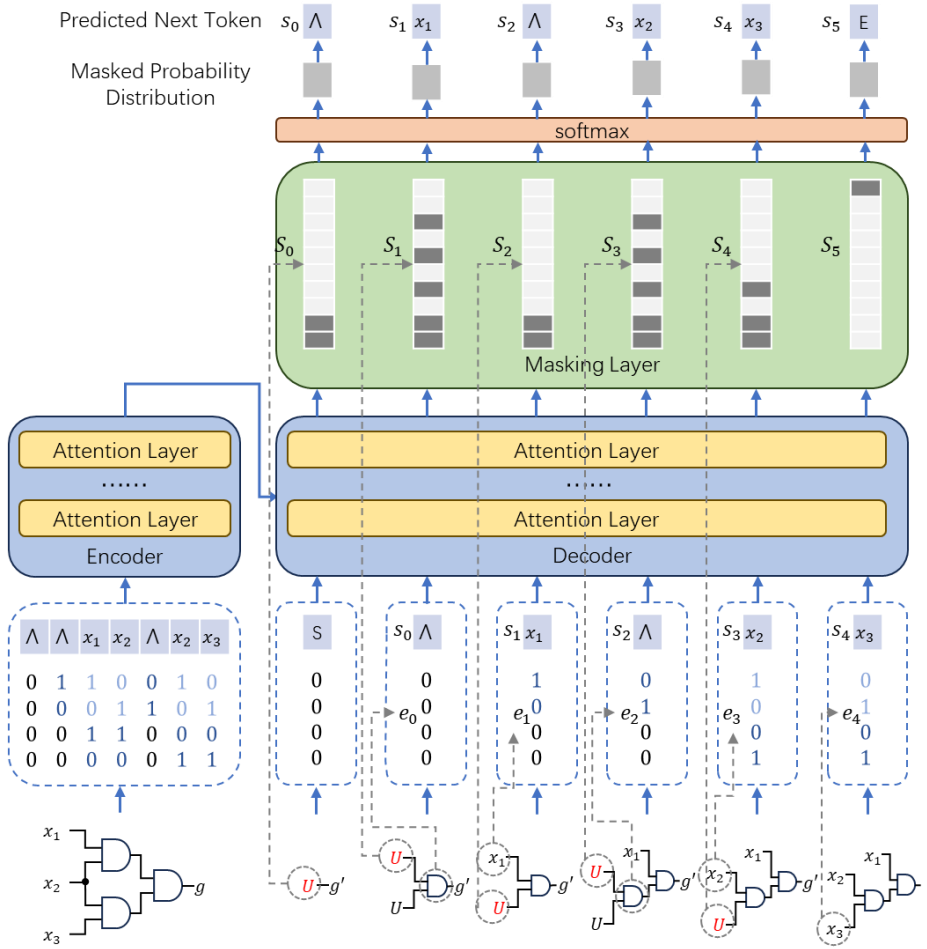
$$S_t = \{x_0, \wedge, \overline{\wedge}\}$$

The Circuit Transformer

- The Boolean function is encoded by the Transformer encoder
- A masking layer is added before the softmax layer of the Transformer decoder
  - Only allows tokens in valid move S_t to have positive possibilities



The Training Stage



The Inference Stage

Experiments

- We trained a Circuit Transformer with 88M parameters on 15M random generated circuits
  - Circuit size: 8-input, 2-output (can specify $1.34 \cdot 10^{154}$ different circuits)
- Task: generate equivalent yet more compact forms of input circuits
- Results
  - Zero violation of equivalence constraints
  - Optimization performance close to ground truth (even better with MCTS enabled)

In distribution ↓ Out of distribution ↓

Methods	Random circuits		IWLS FFWs	
	Unsuccessful cases	Avg. size	Unsuccessful cases	Avg. size
Boolean Chain	5.07% (5.07%)	15.25	11.36% (11.26%)	17.24
Boolean Chain (beam size = 16)	2.16% (2.16%)	14.89	6.34% (6.29%)	17.15
Boolean Chain (beam size = 128)	1.91% (1.91%)	14.87	5.97% (5.94%)	17.15
AIGER	4.32% (4.32%)	15.14	8.35% (7.77%)	17.19
AIGER (beam size = 16)	1.85% (1.85%)	14.87	4.62% (4.37%)	17.12
AIGER (beam size = 128)	1.71% (1.71%)	14.86	4.24% (3.99%)	17.12
Circuit Transformer w/o TPE	2.14% (0%)	15.02	6.63% (0%)	17.33
Circuit Transformer	1.14% (0%)	14.79	4.76% (0%)	17.17
Circuit Transformer ($K = 10$)	0.20% (0%)	14.02	2.83% (0%)	16.92
Circuit Transformer ($K = 100$)	**0.17%** (0%)	**13.73**	**2.63%** (0%)	**16.73**
Resyn2 (ground truth for training)	/	14.56	/	16.82

Better performance with MCTS enabled

Zero violation of equivalence constraints

Thank you!

GitHub: https://github.com/snowkylin/circuit-transformer

Online Demo: https://huggingface.co/spaces/snowkylin/circuit-transformer-demo

Xihan Li

Department of Computer Science, University College London

xihan.li@cs.ucl.ac.uk

https://snowkylin.github.io

ICLR International Conference On Learning Representations