Circuit Transformer: End-to-end Circuit Design by Predicting the Next Gate

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**

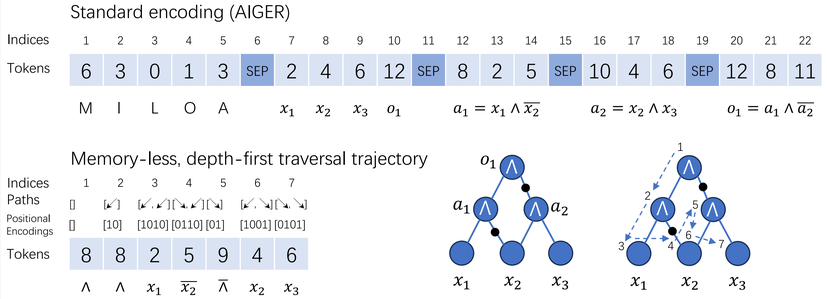
Recent advances in large language models (LLMs) have computationally mastered human language through predictive modeling. Extending this concept to electronic design, we explore the idea of a "circuit model" trained on circuits to predict the next logic gate, addressing structural complexities and equivalence constraints. By encoding circuits as memory-less trajectories and employing equivalence-preserving decoding, our trained "Circuit Transformer" with 88M parameters outperforms existing neural approaches in both feasibility and optimality.

1. Neural Encoding of Circuits

Circuits are "unfolded" as a depth-first traversal trajectory.

Allow redundancy for the ease of neural decoding.

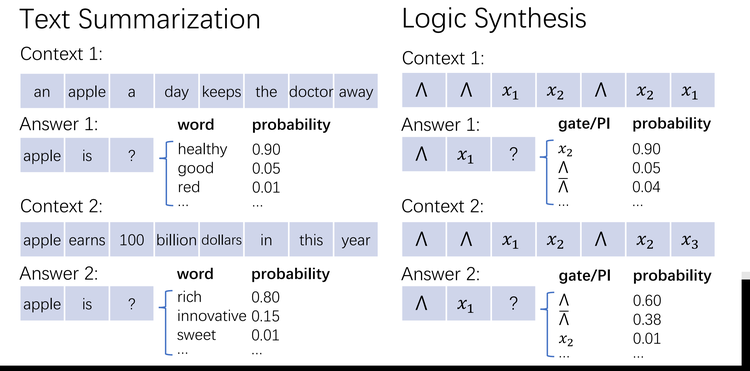
Tree positional encodings to indicate the position of nodes.

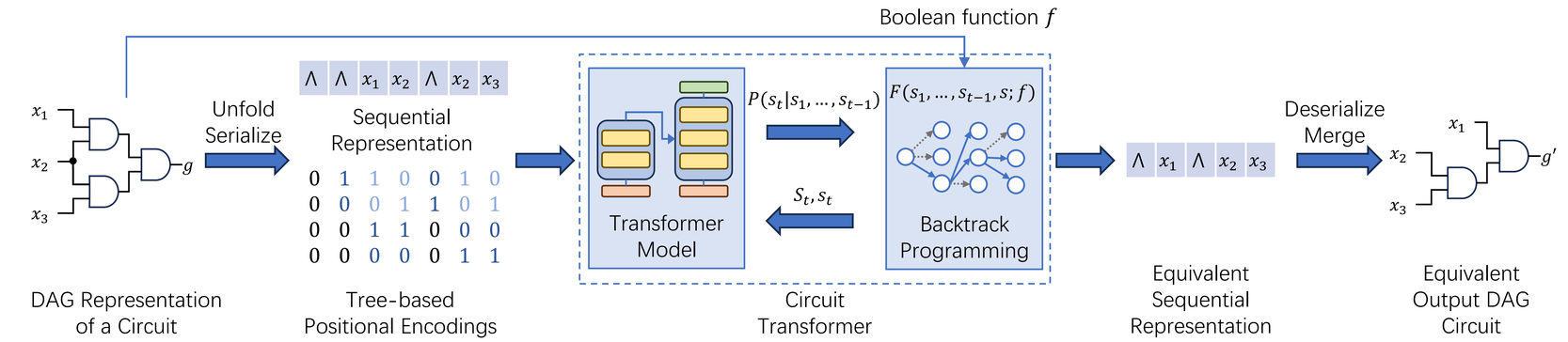


2. Circuit Model: Predict the Next Gate

Circuit model: $P(s_t|s_1, \dots, s_{t-1})$, the probability of each gate or input at step t, given the previous $t-1$ nodes.

Design circuits via simply predicting the next logic gate, just like what large language models do to master human languages!





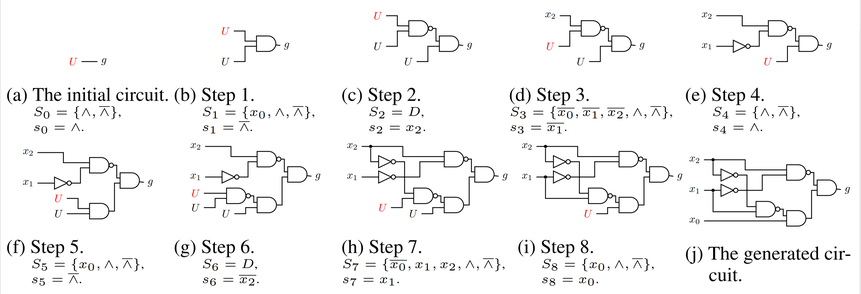
The pipeline of Circuit Transformer that generates a new circuit that is strictly equivalent to an existing one. The backtracking programming module acts as a masking layer at the end of the Transformer decoder.

3. Generating Circuits with Equivalence Preserved

In this work, we pay special attention to step-by-step generation of circuits (with encodings $s_1, \dots, s_n$) that are strictly equivalent to an existing circuit g. To achieve this, we apply backtrack programming in each step t to find feasible candidates $s_t \in S_t$, so that s_t will not result in a violation of equivalence constraints (so the "cutoff properties" $F(s_1, \dots, s_t; g)$ holds). In this way, we remodel the constrained generation problem as

$$\begin{aligned} \textbf{Find} \quad & s_1, \dots, s_n \\ \textbf{s.t.} \quad & F(s_1, \dots, s_t; g) \text{ holds}, t = 1, \dots, n \\ & s_1, \dots, s_n \text{ represents a unique circuit} \end{aligned}$$

We find that such a process can be implemented efficiently with our proposed neural encoding, together with three-valued logic and short-circuit evaluation.



(a) The initial circuit. $S_0 = \{\wedge, \overline{\wedge}\}$, $s_0 = \wedge$.
(b) Step 1. $S_1 = \{x_0, \wedge, \overline{\wedge}\}$, $s_1 = \overline{\wedge}$.
(c) Step 2. $S_2 = D$, $s_2 = x_2$.
(d) Step 3. $S_3 = \{\overline{x_0}, \overline{x_1}, \overline{x_2}, \wedge, \overline{\wedge}\}$, $s_3 = \overline{x_1}$.
(e) Step 4. $S_4 = \{\wedge, \overline{\wedge}\}$, $s_4 = \wedge$.
(f) Step 5. $S_5 = \{x_0, \wedge, \overline{\wedge}\}$, $s_5 = \overline{\wedge}$.
(g) Step 6. $S_6 = D$, $s_6 = \overline{x_2}$.
(h) Step 7. $S_7 = \{\overline{x_0}, x_1, x_2, \wedge, \overline{\wedge}\}$, $s_7 = x_1$.
(i) Step 8. $S_8 = \{x_0, \wedge, \overline{\wedge}\}$, $s_8 = x_0$.
(j) The generated circuit.

4. Experiments

We supervisedly trained a Circuit Transformer on random circuits to solve the circuit size minimization problem, generating equivalent yet more compact forms of input circuits.

Evaluation: both on synthetic circuits and real IWLS benchmarks

Baselines: Two other popular circuit encodings

Methods	Random circuits		IWLS FFWs	
	Unsuccessful cases	Average circuit size	Unsuccessful cases	Average circuit 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

Our proposed approach outperforms existing approaches in both feasibility and optimality.

Scan the QR code for the full paper, poster and future updates. Or visit:

https://snowkylin.github.io/publications

Correspondence email: xihan.li@cs.ucl.ac.uk

Logic Synthesis with Generative Deep Neural Networks

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**

In this work, we introduced the first logic synthesis operator powered by generative deep neural networks. More specifically, a rewriting operator is developed based on the Circuit Transformer model, named `ctrw` (Circuit Transformer Rewriting), which incorporates the following techniques: (1) a two-stage training scheme for the Circuit Transformer tailored for logic synthesis, with iterative improvement of optimality through self-improvement training; (2) integration of the Circuit Transformer with state-of-the-art rewriting techniques to address scalability issues, allowing for guided DAG-aware rewriting. Experimental results on the IWLS 2023 contest benchmark demonstrate the effectiveness of our proposed rewriting methods.

1. Train a Circuit Transformer for Small-sized Logic Synthesis

End-to-end, supervised learning with:

- **Input:** randomly generated circuits with unique canonicalizations (realistic circuits do NOT work well)
- **Output:** optimized circuits with existing optimizers (resyn2)

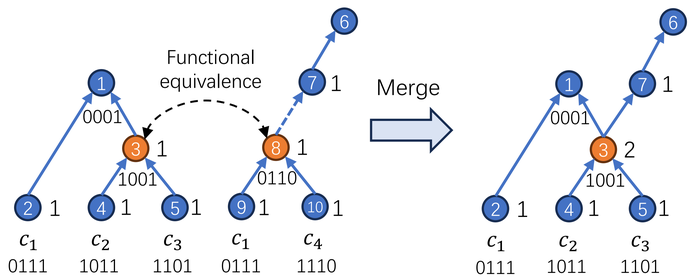
2. Circuit Size Minimization as a Markov Decision Process

We can minimize the number of AND gates of the generated circuit by attaching an immediate reward function $R(s_1, \dots, s_t, s)$ to the generation of token s at step t, so that the generation process can be modelled as an MDP.

- State at step t: $g_1, \dots, g_t$
- Action: g
- Immediate reward: $R(g_1, \dots, g_t, g) = \Delta + \begin{cases} -1, g = \wedge \text{ or } g = \overline{\wedge} \\ 0, \text{otherwise} \end{cases}$

(In which Δ reflects the refinement of equivalent node merging)

- Cumulative reward: negative number of AND nodes in the generated circuits



Step (Node ID)	1	2	3	4	5	6	7	8	9	10
Token	∧	c_1	∧	c_2	c_3	∧	∧	$\overline{\wedge}$	c_1	c_4
Immediate Reward	-1	0	-1	0	0	-1	-1	-1	0	**1**
Cumulative Reward	-1	-1	-2	-2	-2	-3	-4	-5	-5	**-4**

Negative number of AND nodes

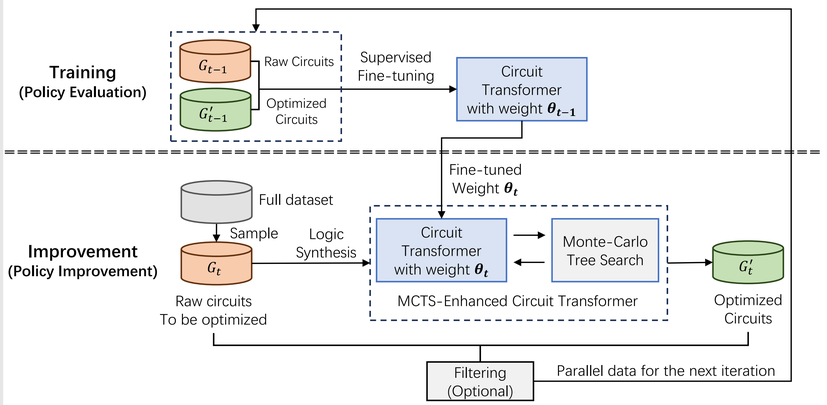
(node 1, 3, 6, 7, 8)

(node 1, 3, 6, 7, node 8 is merged)

3. Iterative Self-Improvement Training

Iteratively fine-tune the model to generate more compact circuits with Monte-Carlo tree search (MCTS) based self-improving.

- **Training stage**: fine-tune the model with parallel data
- **Improvement stage**: generate new parallel data with fine-tuned model and MCTS



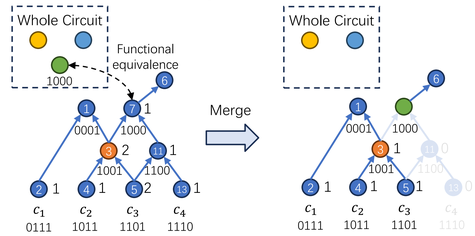
4. Cooperate Circuit Transformer with Fanout-Free Window Rewriting

We leverage the fanout-free window rewriting framework in [2], and apply our trained Circuit Transformer with self-improvement to replace each fanout-free window by a more compact one. MCTS can also be applied during the rewriting. Such replacement enables additional flexibility that the replaced circuit does not require to be equivalent to the sub-circuit it replaces.

[2] https://ieeexplore.ieee.org/document/10247727

5. Guided DAG-aware Rewriting

We refined the immediate reward to reflect the node merging in the rewriting process. Then MCTS is guided by the reward function to minimize the size of final rewritten circuit after node merging.



Step (Node ID)	1	2	3	4	5	6	7	8	9	10	11	12	13
Token	∧	c_1	∧	c_2	c_3	∧	∧	$\overline{\wedge}$	c_1	c_4	∧	c_3	c_4
Immediate Reward	-1	0	-1	0	0	-1	-1	-1	0	1	-1	0	**2**
Cumulative Reward	-1	-1	-2	-2	-2	-3	-4	-5	-5	-4	-5	-5	**-3**

Node 7 is replaced by the green node in the whole circuit. Node 11 is dereferenced after replacement.

6. Experiments

Methods	Avg. Improv.	Time cost
Drw Rewriting (ABC)	15.42%	<0.01s
MFFW Rewriting (in Python)	21.16%	1-300s
Ctrw (w/o self-improvement)	18.55%	1-250s
Ctrw	23.23%	1-285s
Ctrw with MCTS	26.02%	300-31000s
Ctrw with MCTS and guided DAG-aware Rewriting	30.19%	240-35000s

Our proposed approach successfully generated strictly feasible circuits (checked via cec), and demonstrated significant effectiveness in reducing circuit size. (However, the efficiency still needs to be improved, especially for MCTS)

Scan the QR code for the full paper, poster and future updates. Or visit:
https://snowkylin.github.io/publications

Correspondence email: xihan.li@cs.ucl.ac.uk