Circuit Transformer: A Transformer That Preserves Logical Equivalence

Xihan Li[1], Xing Li[2], Lei Chen[2], Xing Zhang[2], Mingxuan Yuan[2], Jun Wang[1]
[1] University College London [2] Huawei Noah's Ark Lab, Hong Kong
{xihan.li,jun.wang}@cs.ucl.ac.uk
{li.xing2,lc.leichen,zhangxing85,Yuan.Mingxuan}@huawei.com

Abstract

Despite the significant advancements in next token prediction models, they are often considered less promising for logic tasks that demand exact precision. In this study, we introduce an end-to-end Transformer model, the "Circuit Transformer", which ensures such exactness by generating new logic circuits that are strictly equivalent to existing ones. This is accomplished through a novel approach that formulates equivalent circuit generation as a constrained sequential generation process for backtrack programming. Then we propose a top-down and sequential circuit representation method with advantageous "cutoff properties", enabling next-token prediction models to generate circuits in a manner similar to natural language generation, while strictly adhering to the feasible region. This equivalence-preserving property also allows optimization methods to explore freely without violating constraints. Experimentally, we trained an 88-million-parameter Circuit Transformer to generate equivalent yet more compact forms of input circuits, outperforming existing neural approaches on both synthetic and real world benchmarks, without any violation of complex equivalence constraints.

1 Introduction

In this work, we focus on generating a proper logic gate implementation g of a Boolean function

$$\boldsymbol{y} = f(\boldsymbol{x}), \tag{1}$$

in which $\boldsymbol{x} = (x_1, \dots, x_N) \in \{0,1\}^N$ is the N-dimensional input, $\boldsymbol{y} = (y_1, \dots, y_M) \in \{0,1\}^M$ is the M-dimensional output. While a Boolean function f may have many different logic gate implementations g, all these implementations are strictly constrained by the logical equivalence. That is defined as $g \in C(f)$, in which

$$C(f) = \{g | g_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \{0,1\}^N, i = 1, \dots, M\} \tag{2}$$

is the feasible region of g under f consisting of $2^N \cdot M$ equality constraints. A logic gate implementation g of a Boolean function is also called a *circuit*. [1] An example is shown in Figure 1.

This equivalent circuit generation problem is highly concerned in the fields of Boolean algebra and computational complexity theory, especially in the context of finding a compact form of a circuit g with minimal number of logic gates (circuit size minimization). That is

$$\mathbf{min}\, |g'| \quad \mathbf{s.t.} \quad g' \in C(g), \tag{3}$$

where $|g'|$ is the number of logic gates[2] in g'. This leads to important research areas such as logic optimization [43], minimum circuit size problem [14, 11] and circuit complexity theory [42, 35].

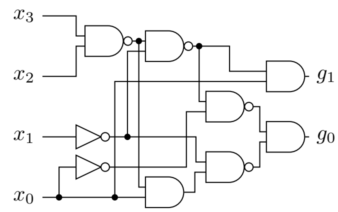
[1] More specifically, combinatorial logic circuit, which is referred to as "circuit" in short in this paper unless otherwise stated.

[2] NOT gates (inverters) are not counted in our paper to align with the mainstream research setting.

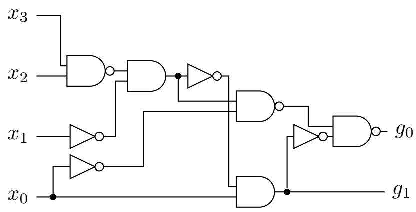
Preprint. Under review.

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

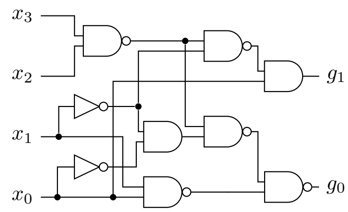
(a) A Boolean function $(y_1, y_0) = f(x_3, x_2, x_1, x_0)$, in which $x_0, x_1, x_2, x_3, y_0, y_1 \in \{0, 1\}$.



(b) An initial feasible implementation of the Boolean function f in Figure 1a, with 7 AND/NAND gates.



(c) A feasible implementation of f with 5 AND/NAND gates, which is optimal in size.



(d) An infeasible implementation. In this circuit $g_0 = 0 \neq y_0$ when $\boldsymbol{x} = (1, 1, 0, 1)$.

Figure 1: An example showing how a Boolean function can be implemented by circuits, i.e., cascade connections of logic gates. and are the AND gate and NOT gate respectively, and is the NAND gate, an AND gate followed by a NOT gate. The circuits shown in Figure 1b and Figure 1c are both *feasible* implementations of f, while the latter is more compact in size. The circuit shown in Figure 1d is *infeasible*, even if the difference is as small as one bit out of 32.

Practically, it also lies in the core of electronic design automation (EDA) in the semiconductor industry, as chips are built upon basic logic gates to fulfill certain Boolean functionalities. A compact circuit allows for more chips to be fabricated on a wafer, or more components to be integrated in a single chip, thereby heavily impacting the cost and performance of final chip products.

However, like many other logical and constrained tasks, symbolic approaches have taken the lead of tackling the aforementioned problem for decades, both in theory and practice. For the circuit size minimization problem, it is typically reduced to a series of Boolean satisfactory (SAT) problems and then solved via SAT solvers [17, 15, 37, 10]. More scalable methods in the EDA industry involve multiple heuristic operators, which are executed sequentially in a pre-defined operator sequence to reduce the size of an initial implementation [3, 23, 53]. While machine learning (ML) approaches emerge in recent years, they tackle sub-problems in aforementioned methods, such as SAT solver acceleration [32, 44, 51, 9] and operator sequence scheduling [48, 12, 8, 52, 46]. Direct generation of feasible circuits in the context of ML is regarded as extremely difficult, due to the complicated constraints and strict feasibility requirement [13, 21, 38, 29].

In this work, we firstly drive next token prediction models — especially Transformer models — to generate strictly feasible circuits while preserving logical equivalence. We model the problem of equivalent circuit generation as a constrained sequential pattern generation process, which can be solved with backtrack programming, a promising framework for constraint satisfaction. Then, we proposed a sequential representation of circuits, with associated "cutoff properties" so as to cooperate with backtrack programming. With top-down specification, three-valued logic and short-circuit evaluation techniques introduced, our proposed cut-off properties for backtrack programming owns beneficial characteristics, allowing smooth inference of circuits analogous to typical natural language generation, while the equivalence constraints are strictly satisfied. With tree-based positional encodings [33], such a representation can also serve as a decent neural encoding of circuits. Finally, a transition approach is proposed to enable tree-based generation method to generate directed acyclic graphs via merging nodes with equivalent functionalities.

Moreover, such an equivalence-preserving representation enables decisional optimization methods to efficiently generate equivalent circuit with desired objective (e.g., size minimization). Without the barrier of constraint violation, equivalent circuit generation can be modelled as a Markov decision process with a fixed action space. The only requirement is to designate an immediate reward function to reflect the optimization goal. An example on circuit size minimization problem is also attached.

To demonstrate the effectiveness of our proposed techniques, we developed an end-to-end approach to tackle the aforementioned circuit size minimization problem. A Transformer model of 88 million

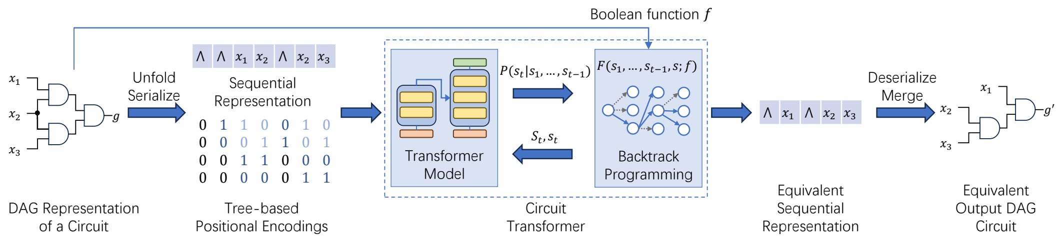


Figure 2: The pipeline of Circuit Transformer that transforms a circuit to a strictly equivalent one. The backtracking programming module acts as a masking layer at the end of the Transformer decoder.

parameters is trained in a supervised way to generate strictly equivalent yet more compact implementations of given circuits. Experimental results show that the trained model is capable of generating strictly equivalent implementations for *all* unseen circuits in the test set, and the size decrease is close to the traditional method that serves as the supervised signal.

To conclude, we make the following contributions:

- A novel formulation of the equivalent circuit generation problem as a sequential pattern generation process for backtrack programming.
- A sequential circuit representation method with beneficial cutoff properties, that allows next token prediction models to generate strictly equivalence-preserving circuits analogous to natural language generation, and can also serve as a neural encoding of circuits.
- A formulation of circuit optimization as a Markov decision process, with an immediate reward function assigned to reflect the optimization goal.
- An end-to-end Transformer model "Circuit Transformer" that allows transformation between logic circuits with strict equivalence preserved, with extensive experiments on the circuit size minimization problem demonstrating its exact feasibility of complex equivalence constraints.

2 Related Work

Recent years have witnessed the tremendous success of next token prediction models, especially Transformer models [41] behind mainstream large language models (LLM). However, due to their probabilistic nature, such models are usually less promising in domains requiring exact preciseness. Therefore, the mainstream paradigm of ML in such domains is to aid traditional methods in solving sub-problems that are more relaxed with respect to exactness. For example, AlphaGeometry [40] trained a Transformer model to aid geometry theorem proving, which requires strict exactness, by suggesting candidate auxiliary points that does not require such exactness. In the context of circuit design, such relaxed sub-problems include SAT solver acceleration [32, 44, 51, 9], circuit representation learning [50, 47, 19, 45], learning based graph optimization [28, 20], operator sequence scheduling [48, 12, 8, 52, 46], and placement and routing [22, 5].

Nonetheless, there are a few research works [31, 30, 6] that attempt to generate circuits or logical expressions directly via next token prediction models. However, none of them takes feasibility guarantee into consideration, and not surprisingly, failure cases do commonly exist in their reported results, which limit their practical usage. In [31], 20% - 70% generated circuits violate the input specification in different datasets, while a follow-up work [30] mitigates the invalid percentage to 16% - 65% with pre-trained language models. In [6], 5% - 10% cases failed to be fully recovered when the number of operators are between 25 and 50. Apart from the low-level circuit generation, many researchers focus on transferring the software code generation capability of LLM to high-level hardware code generation, but these methods still suffer from functional inequivalence [21, 38, 29].

There are also research works that guarantee the feasibility of solutions via action masking, especially for problems involving routing such as maze game, traveling salesman problem, and vehicle routing problem [27, 18, 7]. However, these problems are usually intuitive to be sequentially modelled, allowing action masks to simply filter invalid actions such as wall-hitting directions and visited cities. Action masking techniques under complicated constraints have yet to be explored.

For sequential representation of gate-level circuits, existing formats include graph-based ones like AIGER [2] and BLIF [1], and code-based ones like Verilog [39] and VHDL [36]. The former ones specify a logic circuit by listing all the gates with their connection to child nodes, which can be regarded as a specialization of adjacency list to represent a graph. The latter ones are more general hardware description languages that are close to programming languages in format. However, they lack essential characteristics that allow cutoff properties to be defined under the framework of backtrack programming.

3 Methods

To ensure that next-token prediction models consistently produce circuits that meet all $2^N \cdot M$ equivalence constraints specified in Equation 2, we integrate these models with a backtracking programming framework. We then introduce a novel sequential representation of circuits designed to work efficiently within this framework. Additionally, we propose a Markov decision process formulation to optimize the generated circuits while maintaining equivalence.

3.1 Equivalent Circuit Generation with Backtrack Programming

Next token prediction targets *sequential generation*, that is, a series of tokens $s_1, s_2, \ldots$ is sampled from a probability distribution $P(s_t|s_1, \ldots, s_{t-1})$ in a recursive manner. To generate sequences $s_1, \ldots, s_n$ with strict constraints to make sure that certain property $F(s_1, \ldots, s_n) \in \{0, 1\}$ holds, a promising approach is backtrack programming [16]. It involves the invention of intermediate "cutoff" properties $F(s_1, \ldots, s_t)$ for $1 \leq t < n$, which have the following two characteristics

Characteristic 3.1 (Inheritability). *$F(s_1, \ldots, s_t)$ is true whenever $F(s_1, \ldots, s_{t+1})$ is true;*

Characteristic 3.2 (Incrementality). *$F(s_1, \ldots, s_t)$ is fairly easy to test, if $F(s_1, \ldots, s_{t-1})$ holds.*

Proper cutoff properties allow candidate sequences to be incrementally built towards the solution, and backtrack when a candidate cannot possibly be completed to a feasible solution. Given partial sequence $s_1, \ldots, s_t$ at time step t that $F(s_1, \ldots, s_t)$ holds, the feasibility of the next token s_{t+1} can be tested quickly via $F(s_1, \ldots, s_t, s_{t+1})$, and backtrack can occur when all s_{t+1} has been recursively explored. A detailed introduction of the framework is leaved in the appendix.

Following this direction, to leverage next token prediction to generate strictly equivalent circuits, we aim to find a proper sequential representation of circuits, as well as proper cutoff properties in each step to indicate the equivalence, so that we can remodel the equivalent circuit generation problem

$$\textbf{find} \quad g \quad \textbf{s.t.} \quad g \in C(f) \tag{4}$$

as a constrained generation process of sequential patterns

$$\begin{aligned} \textbf{find} \quad & s_1, s_2, \ldots, s_n \\ \textbf{s.t.} \quad & F(s_1, \ldots, s_t; f) \text{ holds}, t = 1, \ldots, n \\ & s_1, \ldots, s_n \text{ represents a unique circuit} \end{aligned} \tag{5}$$

so as to apply backtrack programming to guarantee the feasibility. In Equation 5, $s_1, \ldots, s_t$ is a series of sequential patterns and represents a class of circuits, which will be elaborated in the next section. $F(s_1, \ldots, s_t; f)$ is the cutoff property in step t, which holds if and only if the circuits represented by $s_1, \ldots, s_t$ are all within the feasible region $C(f)$.

In such a way, given a next token prediction model

$$P(s_t|s_1, \ldots, s_{t-1}) \tag{6}$$

that provides the probability distribution of s_t given the previous tokens $s_1, \ldots, s_{t-1}$, we can generate the circuit by Algorithm 1, which is an adaptation of the backtrack programming framework introduced in [16] with candidate tokens ordered by the probability distribution.

3.2 A Sequential Representation of Circuits with Cutoff Properties

With the new formulation in Equation 5 and Algorithm 1, the circuit generation problem reduces to finding a sequential representation $s_1, s_2, \ldots$ of circuits, with corresponding cutoff properties $F(s_1, \ldots, s_t; f), 1 \leq t < n$ satisfying Characteristic 3.1 and Characteristic 3.2 that keep the sequential representation within the equivalence class of f. Moreover, as our goal is to generate *one* rather than *all* feasible circuits, the most efficient circumstance in Algorithm 1 is that the "backtracking" process in line 11-14 is never executed. That is, $S_t \neq \emptyset$ all the time. Thus, it is desirable for an efficient representation to guarantee this, avoiding any compulsory backtracking when only a single circuit is required to be delivered. That is,

Algorithm 1 Equivalent Circuit generation with Backtrack Programming

Input: The Boolean function f that the generated circuit should be equivalent to. Next token prediction model $P(s_t|s_1, \ldots, s_{t-1})$.
Output: A feasible circuit g satisfying $g \in C(f)$.
1: $t \leftarrow 1$
2: **while** true **do**
3:     Compute a probability distribution of $s_t \in D$ by the next token prediction model

$$p_t \leftarrow P(s_t|s_1, \ldots, s_{t-1})$$

4:     Set $S_t \leftarrow \{s \in D|F(s_1, \ldots, s_{t-1}, s; f) \text{ holds}\}$
5:     **while** true **do**
6:         **if** $S_t \neq \emptyset$ **then**
7:             $s_t \leftarrow \arg\max_{s \in S_t} p_t(s)$
8:             **if** $s_1, \ldots, s_t$ represents a unique circuit **then return** the corresponding circuit
9:             $t \leftarrow t + 1$
10:             **break**
11:         **else**   ▷ The backtrack process. If Characteristic 3.3 holds, this branch will never be executed.
12:             $t \leftarrow t - 1$
13:             $S_t \leftarrow S_t \backslash s_t$
14:         **end if**
15:     **end while**
16: **end while**

a	**¬a**
1	0
0	1
U	*U*

(a) NOT operator

a ∧ b		**a**: **1**	**0**	**U**
b	**1**	1	0	*U*
	0	0	0	0
	U	*U*	0	*U*

(b) AND operator

a ∨ b		**a**: **1**	**0**	**U**
b	**1**	1	1	1
	0	1	0	*U*
	U	1	*U*	*U*

(c) OR operator

a ≃ b		**a**: **1**	**0**	**U**
b	**1**	1	0	1
	0	0	1	1
	U	1	1	1

(d) SIMEQ operator

Table 1: The truth tables for NOT, AND, OR and SIMEQ operators in three-valued logic.

Characteristic 3.3 (Backtrack Elimination)**.** *$S_t \neq \emptyset$ is always guaranteed in line 4 of Algorithm 1.*

In this way, the next-token prediction process can always proceed forward efficiently, analogous to typical natural language generation.

For tasks requiring exploration of feasible region for circuits, it is important for a representation of circuits to cover the widest possible (ideally all) feasible circuits, minimizing the miss of targets due to the restriction of representation. For example, while we can always generate a strict equivalent circuit for a Boolean function f via sum-of-product or product-of-sum forms, such forms are too restricted for any feasible region exploration. Therefore, we have the following desired characteristic:

Characteristic 3.4 (Completeness)**.** *For all $g \in C(f)$, there exists a sequence $s_1, \ldots, s_n$ that uniquely represents g.*

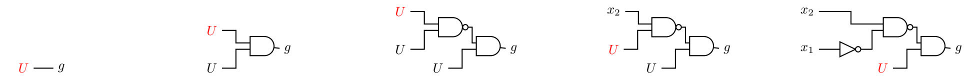
Now we propose a sequential representation of circuits with cutoff properties, that owns all the aforementioned characteristics.

First, while circuits are usually built in a bottom-up manner from inputs to outputs, we notice that the equivalence constraints are applied on each *output* of the circuit. That is, given the index of output i, a constraint $f_i(\boldsymbol{x}) = g_i(\boldsymbol{x}), \forall \boldsymbol{x} \in \{1, 0\}^N$ is only possible to be validated when the corresponding circuit output g_i has been built. Therefore, we adopt a special top-down order, specifying a circuit from outputs to inputs, to allow constraint validation throughout the construction process.

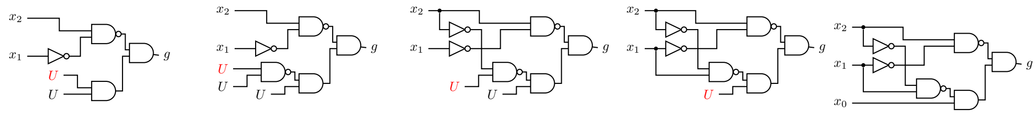
Then, to allow indeterminacy in the circuit representation, we include the three-valued logic into the circuit evaluation process. That is, besides $\{0, 1\}$ which indicate *false* and *true*, there is another truth value "U" which means *unknown*. The truth tables of such logic for NOT, AND and OR operators are shown in Table 1. Additionally, we define a binary operator "SIMEQ" ($\simeq$, is similar or equal to), which is equivalent to the equal operator ($=$) for $\{0, 1\}$, while accommodating U by $U \simeq 0$ and $U \simeq 1$. During the generation process, we relax the feasible region from Equation 2 to

$$C'(f) = \{g|g_i(\boldsymbol{x}) \simeq f_i(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \{0, 1\}^N, i = 1, \ldots, M\} \tag{7}$$

so that the occurrence of U in the output will not violate the constraint. For simplicity, here we assume $M = 1$ and leave multi-output cases to be discussed later. The generated circuit g is initialized to be



(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.

Figure 3: An example showing how a strictly feasible circuit can be built with our proposed sequential representation with cutoff properties. $f(x_2, x_1, x_0)$ is a Boolean function with $f(0,1,0) = f(1,1,1) = 1$ and $f(x_2, x_1, x_0) = 0$ otherwise. $D = \{x_0, \overline{x_0}, x_1, \overline{x_1}, x_2, \overline{x_2}, \wedge, \overline{\wedge}\}$, U denotes the wildcard node, the next wildcard node to be replaced is marked in red. $S_t = \{s \in D | F(s_1, \ldots, s_{t-1}, s; f) \text{ holds}\}$.

a single constant node U, which we call "wildcard node" as it can potentially represent any feasible circuits. So initially, $g(\boldsymbol{x}) \equiv U$ no matter what the input $\boldsymbol{x}$ is. This is within the relaxed feasible region in Equation 7 but provides no information about the circuit structure.

Given the initial circuit, the sequential generation process acts as refining the circuit g by recursively replacing a wildcard node to a specific one $s_t \in D$, which can be either a new logic gate or a new primary input. For a new logic gate, all its inputs will be initialized to wildcard nodes that need further refinement. With the proposed top-down approach, the values of $g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^M$ in Equation 7 can always be evaluated throughout the construction process, following the truth tables in Table 1. Note that the introduction of three-valued logic also enables short-circuit evaluation with unknown values. For an AND gate $c(\boldsymbol{x}) = a(\boldsymbol{x}) \wedge b(\boldsymbol{x})$, if one of the inputs ($a(\boldsymbol{x})$ or $b(\boldsymbol{x})$) is evaluated to be 0 given specific $\boldsymbol{x}$, then $c(\boldsymbol{x}) = 0$ no matter what the other input is evaluated, even if it is U. The same logic applies for the OR gate when one of the inputs is evaluated to be 1.

Then, the cutoff properties $F(s_1, \ldots, s_t; f)$ holds if and only if for all $\boldsymbol{x} \in \{0,1\}^M$, the output of the constructed circuit $g^{(t)}$ given input $\boldsymbol{x}$ at time step t is similar or equal to $f(\boldsymbol{x})$. That is,

$$g^{(t)}(\boldsymbol{x}) \simeq f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \{0,1\}^M \tag{8}$$

or simply, $g^{(t)} \in C'(f)$.

For the ending criteria, as a wildcard node can potentially be any circuit, only when all the wildcard nodes are recursively replaced by specific ones, can the sequence uniquely represent a circuit, which marks the end of the generation process in line 8 of Algorithm 1. For the order of replacement when multiple wildcard nodes exist, we follow a fixed order that prioritizes those with the largest distance from the output and the left child of a gate over the right one. For multi-output cases, we generate the circuit for each output separately, and combine them together via node merging which will be discussed in the next section.

For the selection of logic gates (vocabulary list) in the sequential representation, we note that the combination of AND and NOT gates can express all possible truth tables of Boolean functions (which is termed "functional completeness"), and is also commonly adopted in practical circuit representation [2]. Therefore, we adopted this setting, with an alteration that we merged the NOT gate with the AND gate and primary inputs. Instead of assigning the NOT gate an individual token, each primary inputs and the AND gate has two versions: the original ones ($x_1, \ldots, x_N, \wedge$) and the inverse ones ($\overline{x_1}, \ldots, \overline{x_N}, \overline{\wedge}$), so the vocabulary list contains $2N + 2$ tokens in total[3]. This allows us to significantly shorten the sequence with a moderate increase of vocabulary size.

An example of our proposed representation and cutoff properties are shown in Figure 3. We leave the proof of Characteristic 3.1, 3.2, 3.3 and 3.4 in the appendix.

3.3 From Trees to Directed Acyclic Graphs

In the last section, we proposed a sequential representation of circuits based on recursive replacement of wildcard nodes in the top-down manner. Such an approach implicitly assumes that a logic gate

[3]This does not include special tokens such as [EOS] and [PAD] in Transformer models.

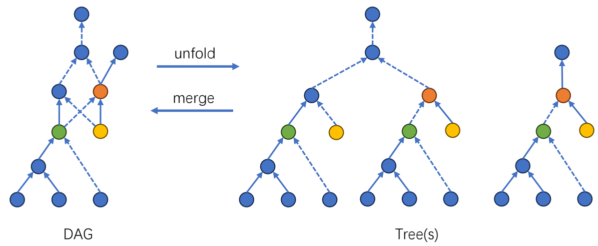


Figure 4: Transition between a DAG and one or more trees. The shown DAG is an abstraction of Figure 1c in which circles represent AND gates, primary inputs and outputs, solid arrows represent wires from outputs to inputs, and dashed arrows represent wires with a NOT gate between the input and the output.

would always have one fan-out[4], restricting the generated circuits to be highly hierarchical with tree structures. However, multi-fanout gates do commonly exist in real-world logic circuits, which shape circuits as directed acyclic graphs (DAGs).

In this section, we show how we extend our method to generate DAG circuits. We notice that a DAG can be "unfolded" to one or more trees once we duplicate every node with outdegree larger than one, so that every node has at most one outgoing edge. For example, the orange node in the left DAG of Figure 4 is duplicated into two individual nodes, where its two outgoing edges are assigned respectively. Reversely, one or more trees can also be transformed to a DAG by merging nodes with structural equivalence. For circuits, such a bidirectional transition will not change the Boolean function it represents. Therefore, we generate a DAG circuit by firstly generating its unfolded tree representation, and then merging equivalent nodes in the generated tree representation. In logic circuit design, different nodes can be not only structurally equivalent but also functionally equivalent [26], which means that their outputs represent the same Boolean functionality. Functionally equivalent nodes can thus be merged as a single node even if they have different underlying structures. Our approach mainly leverages the functional equivalence.

3.4 Neural Encoding of Circuits and Circuit Transformer

While we can deserialize our proposed sequential representation to a DAG circuit via node merging, we can also serialize a given DAG circuit to our proposed sequential representation via a depth-first traversal with node duplication. Given a DAG circuit, we start a traversal from each of its primary outputs, and visit each connected gate in a depth-first, recursive manner. Backtracking occurs when a primary input is reached. Importantly, such a traversal is memory-less, i.e., visited nodes will not be labelled during the traversal, thus a node will appear multiple times in the trajectory if its fan-out is larger than one, corresponding to the node duplication in the last section. When the process is finished, the traversal trajectory $s_1, \ldots, s_n$ is the sequential representation of the unfolded tree version of the original DAG circuit. Note that such an unfolding process may lead to long sequences, especially for nodes with large number of fan-outs. A more compact representation is leaved as future work.

For Transformer models to process the sequential representation, it is important to provide an efficient positional encoding for each node to indicate its position in the circuit. In this work, we utilize the path from the primary output to a given node to indicate the node's position. To achieve this, we follow [33] that encodes the path as a stack of one-hot encodings ("10" for the first input and "01" for the second input). More details are leaved in the appendix.

With all the circuit encoding and generation techniques introduced above, we propose Circuit Transformer, an end-to-end Transformer model that generates a new and functionally equivalent circuit of the input one. The Transformer uses an encoder-decoder architecture, and the original circuit is serialized to our proposed sequential representation, acting as the input of the Transformer encoder. The Transformer's decoding process cooperates with the backtrack programming framework in Algorithm 1 with our proposed cutoff properties in Section 3.2. Given Characteristic 3.3, the backtrack process in line 11-14 will not be actually executed, so S_t in line 4 of Algorithm 1 reduces to a masking layer at the end of the Transformer decoder, filtering invalid tokens by assigning probability zero to them. Finally, node merging proposed in Section 3.3 is applied to the decoded sequence to transform the unfolded tree representation to a DAG circuit. The pipeline is shown in Figure 2.

[4]The fan-out of a gate is the number of inputs driven by the output of the gate.

3.5 Equivalent Circuit Generation as a Markov decision process

An important application of equivalent circuit generation is to optimize circuits with respect to certain objectives. For example, in semiconductor industry, it is of vital importance to reduce the size of a given circuit measured by the number of logic gates, while the specification of the circuit (i.e., the Boolean function it represents) keeps equivalent. Under our proposed sequential representation, it can be achieved by attaching an immediate reward function $R(s_1, \ldots, s_t, s)$ to the generation of token s at step t, so that the sum of the reward function throughout the generation reflects the objective. In this way, the generation process can be regarded as a deterministic Markov decision process, in which the state at step t is the generated tokens $s_1, \ldots, s_t$, the set of actions available from the state is $S_t = \{s \in D | F(s_1, \ldots, s_t, s; f) \text{ holds}\}$, the immediate reward is $R(s_1, \ldots, s_t, s)$, and the next state is $s_1, \ldots, s_t, s$ with probability 1. The process terminates once $s_1, \ldots, s_t$ represents a unique circuit with all wildcard nodes replaced. Such a formulation owns two key advantages. First, the feasibility of the generated circuit is guaranteed once the process terminates. No effort is required to penalize infeasible cases via crafting the reward function. Second, the size of the available action set is at most $2N + 2$, in which N, the number of inputs, is usually moderately small in practice. Other circuit representations typically assign each logic gate a unique ID to describe their adjacency, requiring the size of available actions to be proportional to the number of gates, which is usually significantly larger than N.

For the circuit size minimization problem in Equation 3, the immediate reward function can be defined as

$$R(s_1, \ldots, s_t, s) = \Delta + \begin{cases} -1, & s = \wedge \text{ or } s = \overline{\wedge} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

in which Δ reflects the refinement due to equivalent node merging. Given a depth-first replacement order of wildcard nodes in Section 3.2, the node merging process in Section 3.3 can be done simultaneously with the generation process, whose detail is leaved in the appendix.

4 Experiments

In this section, we supervisedly train a Circuit Transformer to solve the circuit size minimization problem in Equation 3, generating equivalent yet more compact forms of input circuits, and conduct extensive experiments on both synthetic and real-world datasets to evaluate its feasibility and optimality.

The details of the Circuit Transformer model are as follows. We employed an encoder-decoder architecture following [41], each with 12 attention layers. The embedding width and the size of feedforward layer are set as 512 and 2048 respectively, leading to 88,225,800 total parameters. The vocabulary size is 20 (8 inputs and the AND gate, with their inverse, plus [EOS] and [PAD]). Batch size is set to be 128. The maximum length of the input and output sequence is set to be 200. To evaluate the effectiveness of tree positional encoding (TPE) in Section 3.4, we trained Circuit Transformers with and without TPE. The maximal depth of tree positional embeddings is set to be 32. The implementation is based on [49].

We also trained two baseline Transformer models with exactly the same experimental settings, except employing different sequential representation of circuits as follows:

- Boolean Chain [15]: a representation that is extensively applied in SAT-based optimization techniques. A chain is initialized by all the primary inputs of the circuit, and each gate is represented by two prior indices in the chain, indicating the source of its two inputs.
- AIGER [2]: a popular representation of logic circuits with AND and NOT gates. We follow the tokenization setting in [31], representing an AND gate by three tokens followed by a special new line token.

To train and evaluate the Circuit Transformer model, we build a large dataset containing 15 million randomly generated 8-input, 2-output circuits. The supervised signals (i.e., the equivalent circuits that are optimized in size) are generated by the Resyn2 command in ABC [3], a representative optimization flow for circuit size minimization. The detail of random circuit generation is leaved in the appendix. 89% of the data is for training, 1% is for validation and 10% is reserved for testing. All the Transformer models are trained on the training set sufficiently for 5 epochs.

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

Table 2: Results on 10,240 randomly generated circuits, and 10,240 fanout-free windows randomly sampled from the IWLS 2023 benchmark. For unsuccessful cases, the percentage in the bracket corresponds to failures due to equivalence constraint violation. All results of Circuit Transformers show zero violation of equivalence constraints. K denotes the total number of playouts in Monte-Carlo tree search. All the models are supervisedly trained on the Resyn2 optimized circuits.

We employ both a synthetic dataset and a real EDA benchmarks to evaluate the performance of the models:

- Random circuits: 10,240 circuits are randomly sampled from the test set of the aforementioned randomly generated dataset.
- IWLS FFWs: we transform the IWLS 2023 benchmark [24] into circuits represented by AND and NOT gates by the script suggested in [25], and extract 1.5 million 8-input, 2-output fanout-free windows (FFWs) [54]. Then we randomly sample 10,240 circuits from the extracted FFWs.

The latter is of significance in practice, as sub-circuit optimization acts as the core technique of large circuit optimization.

To enhance the performance of the Transformer models in comparison, two heuristics search techniques are applied. Beam search is applied for Transformer models on Boolean chain and AIGER represented circuits. Monte-Carlo tree search is applied for Circuit Transformer to evaluate our proposed MDP formulation in Section 3.5. The detail is leaved in the appendix.

The results are presented in Table 2. On both synthetic and real datasets, the Circuit Transformer surpasses two other Transformer models in terms of feasibility (measured by the percentage of unsuccessful cases) and optimality (measured by the average circuit size). The two baseline models generate unsuccessful circuits for various reasons, including equivalence constraint violations, cycles in circuits, or incomplete specifications, with the most common issue being that the generated circuit is complete and valid but not strictly equivalent to the original. In contrast, the Circuit Transformer's exact precision is empirically shown by zero violation of complex equivalence constraints. The only reason for unsuccessful cases is that wildcard nodes are not fully replaced within the given maximum sequence length of 200. Regarding heuristic search enhancement, while beam search significantly improved feasibility, consistent with findings in [31], the issue of non-equivalence remained prevalent. Conversely, Monte-Carlo tree search in the Circuit Transformer not only substantially reduced unsuccessful cases but also significantly improved the average circuit size, sometimes producing circuits more compact than the ground truth with a moderate number of playouts.

5 Conclusion

In this work, we make an important advancement towards achieving precise generative AI for logic tasks, demonstrating that complex hard-constraint satisfaction is attainable for next-token prediction models when a proper formulation of the constrained problem is established. Inspired by the backtrack programming framework, we introduce a novel approach to the fundamental problem of equivalent circuit generation, enabling next-token prediction models to generate new logic circuits while strictly adhering to complex equivalence constraints. This methodology has the potential to be extended to other fundamental constrained problems, making it a promising area for further research.

References

[1] UoC Berkeley. Berkeley logic interchange format (blif). *Oct Tools Distribution*, 2:197–247, 1992.

[2] Armin Biere. The AIGER And-Inverter Graph (AIG) format version 20071012. Technical Report 07/1, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2007.

[3] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *Computer Aided Verification*, pages 24–40, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[4] D. Chai and A. Kuehlmann. Building a Better Boolean Matcher and Symmetry Detector. In *Proceedings of the Design Automation & Test in Europe Conference*, volume 1, pages 1–6, March 2006. ISSN: 1558-1101.

[5] Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems*, 35:26350–26362, 2022.

[6] Stéphane d'Ascoli, Samy Bengio, Josh Susskind, and Emmanuel Abbé. Boolformer: Symbolic Regression of Logic Functions with Transformers, September 2023.

[7] Lu Duan, Yang Zhan, Haoyuan Hu, Yu Gong, Jiangwen Wei, Xiaodong Zhang, and Yinghui Xu. Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3054–3063, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Antoine Grosnit, Cedric Malherbe, Rasul Tutunov, Xingchen Wan, Jun Wang, and Haitham Bou Ammar. Boils: Bayesian optimisation for logic synthesis. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1193–1196, 2022.

[9] Wenxuan Guo, Hui-Ling Zhen, Xijun Li, Wanqian Luo, Mingxuan Yuan, Yaohui Jin, and Junchi Yan. Machine Learning Methods in Solving the Boolean Satisfiability Problem. *Machine Intelligence Research*, 20(5):640–655, October 2023.

[10] Winston Haaswijk, Mathias Soeken, Alan Mishchenko, and Giovanni De Micheli. SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(4):871–884, April 2020.

[11] John M. Hitchcock and A. Pavan. On the NP-Completeness of the Minimum Circuit Size Problem. 2015.

[12] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 581–586, 2020.

[13] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), jun 2021.

[14] Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 73–79, Portland Oregon USA, May 2000. ACM.

[15] Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015.

[16] Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 5: Mathematical Preliminaries Redux; Introduction to Backtracking; Dancing Links*. Addison-Wesley Professional, 2020.

[17] Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. Finding Efficient Circuits Using SAT-Solvers. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 32–44, Berlin, Heidelberg, 2009. Springer.

[18] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[19] Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. Deepgate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 667–672, 2022.

[20] Xing Li, Lei Chen, Jiantang Zhang, Shuang Wen, Weihua Sheng, Yu Huang, and Mingxuan Yuan. Effisyn: Efficient logic synthesis with dynamic scoring and pruning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[21] Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Verilogeval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2023.

[22] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

[23] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 532–535, July 2006. ISSN: 0738-100X.

[24] Alan Mishchenko. alanminko/iwls2023-ls-contest: Problems and Results of IWLS 2023 Programming Contest, 2023.

[25] Alan Mishchenko and Satrajit Chatterjee. IWLS 2022 Programming Contest, 2022.

[26] Alan Mishchenko, Satrajit Chatterjee, Roland Jiang, and Robert K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, ERL Technical Report, 2005.

[27] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. Reinforcement Learning for Solving the Vehicle Routing Problem. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[28] Walter Lau Neto, Matheus T Moreira, Yingjie Li, Luca Amarù, Cunxi Yu, and Pierre-Emmanuel Gaillardon. Slap: A supervised learning approach for priority cuts technology mapping. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 859–864. IEEE, 2021.

[29] Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: Controlled verilog generation with discriminative guidance. *arXiv preprint arXiv:2402.03375*, 2024.

[30] Frederik Schmitt, Matthias Cosler, and Bernd Finkbeiner. Neural circuit synthesis with pre-trained language models. In *First International Workshop on Deep Learning-aided Verification*, 2023.

[31] Frederik Schmitt, Christopher Hahn, Markus N Rabe, and Bernd Finkbeiner. Neural circuit synthesis from specification patterns. In *Advances in Neural Information Processing Systems*, volume 34, pages 15408–15420. Curran Associates, Inc., 2021.

[32] Daniel Selsam and Nikolaj Bjørner. Guiding High-Performance SAT Solvers with Unsat-Core Predictions. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing – SAT 2019*, pages 336–353, Cham, 2019. Springer International Publishing.

[33] Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[34] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[35] Michael Sipser. *Introduction to the theory of computation*. Cengage Learning, Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States, third edition, international edition edition, 2013.

[36] Kevin Skahill. *VHDL for programmable logic*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[37] Mathias Soeken, Winston Haaswijk, Eleonora Testa, Alan Mishchenko, Luca G. Amaru, Robert K. Brayton, and Giovanni De Micheli. Practical exact synthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 309–314, Dresden, Germany, March 2018. IEEE.

[38] Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):1–31, 2024.

[39] Donald Thomas and Philip Moorby. *The Verilog® hardware description language*. Springer Science & Business Media, 2008.

[40] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, January 2024. Number: 7995 Publisher: Nature Publishing Group.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[42] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999.

[43] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Cheng. *Electronic design automation: synthesis, verification, and test*. The Morgan Kaufmann series in systems on silicon. Morgan Kaufmann/Elsevier, Amsterdam Boston, 2009.

[44] Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. NeuroBack: Improving CDCL SAT Solving using Graph Neural Networks, November 2023. arXiv:2110.14053 [cs].

[45] Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Bei Yu, and Yu Huang. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 61–66, 2022.

[46] Chenghao Yang, Yinshui Xia, Zhufei Chu, and Xiaojing Zha. Logic synthesis optimization sequence tuning using rl-based lstm and graph isomorphism network. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(8):3600–3604, 2022.

[47] Shuwen Yang, Zhihao Yang, Dong Li, Yingxueff Zhang, Zhanguang Zhang, Guojie Song, and Jianye Hao. Versatile multi-stage graph neural network for circuit representation. *Advances in Neural Information Processing Systems*, 35:20313–20324, 2022.

[48] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[49] Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li. TensorFlow Model Garden. `https://github.com/tensorflow/models`, 2020.

[50] Guo Zhang, Hao He, and Dina Katabi. Circuit-gnn: Graph neural networks for distributed circuit design. In *International conference on machine learning*, pages 7364–7373. PMLR, 2019.

[51] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. NLocalSAT: Boosting Local Search with Solution Prediction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1177–1183, Yokohama, Japan, July 2020. International Joint Conferences on Artificial Intelligence Organization.

[52] Keren Zhu, Mingjie Liu, Hao Chen, Zheng Zhao, and David Z. Pan. Exploring logic optimizations with reinforcement learning and graph convolutional network. In *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, pages 145–150, 2020.

[53] Xuliang Zhu, Ruofei Tang, Lei Chen, Xing Li, Xin Huang, Mingxuan Yuan, Weihua Sheng, and Jianliang Xu. A database dependent framework for k-input maximum fanout-free window rewriting. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[54] Xuliang Zhu, Ruofei Tang, Lei Chen, Xing Li, Xin Huang, Mingxuan Yuan, Weihua Sheng, and Jianliang Xu. A database dependent framework for k-input maximum fanout-free window rewriting. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

A Appendix / supplemental material

A.1 Detail of Backtrack Programming Framework

The basic backtrack algorithm is as follows, extracted from [16]:

Given domain D and properties $F(s_1, \dots, s_t)$, this algorithm visites all sequence $s_1, s_2, \dots, s_n$ that satisfy $F(s_1, \dots, s_n)$:

Step 1 [Initialize] Set $t \leftarrow 1$, and initialize the data structures needed later.
Step 2 [Enter level t] (Now $F(s_1, \dots, s_{t-1})$ holds.) If $t > n$, visit $s_1, \dots, s_n$ and go to Step 5, Otherwise set $s_t \leftarrow \min D$, the smallest element of D.
Step 3 [Try s_t] If $F(s_1, \dots, s_t)$ holds, update the data structures to facilitate testing $F(s_1, \dots, s_t, s_{t+1})$, set $t \leftarrow t + 1$, and go to Step 2.
Step 4 [Try again] If $s_t \neq \max D$, set s_t to the next larger element of D and return to Step 3.
Step 5 [Backtrack] Set $t \leftarrow t - 1$. If $t > 0$, downdate the data structures by undoing the changes recently made in Step 3, and return to Step 4. (Otherwise stop.)

The following example shows how cutoff properties can be invented for a specific problem to be solved by backtrack programming. To generate all permutations $s_1, \dots, s_n$ of $\{1, \dots, n\}$ in which all numbers should appear exactly once, we can set $s_i \in \{1, \dots, n\}, i = 1, \dots, n$, and the cutoff property $F(s_1, \dots, s_t)$ holds at step t if and only if $s_i \neq s_j, \forall 1 \leq i < j \leq t$. In such a way, Characteristic 3.1 is obvious, and Characteristic 3.2 also holds, as we only need to test whether $s_i \neq s_t$ holds for all $1 \leq i < t$ given $F(s_1, \dots, s_{t-1})$ holds.

We refer to Section 7.2.2 of [16] for more details about backtrack programming.

A.2 Proof of Characteristics

In this section, we demonstrate that our proposed sequential representation owns Characteristic 3.1, 3.2, 3.3 and 3.4.

Characteristic 3.1: When $F(s_1, \dots, s_{t+1}; f)$ is true, the transition from $s_1, \dots, s_{t+1}$ to $s_1, \dots, s_t$ corresponds to reversely replacing a specific node s_{t+1} with a wildcard node. such a replacement will never break the feasibility, because (1) a wildcard node only represents feasible circuits; (2) the wildcard node at least have a feasible choice to be set as s_{t+1} as $s_1, \dots, s_{t+1}$ is feasible.

Characteristic 3.2: During the generation from step 1 to step $t - 1$, a caching mechanism can be employed to cache the truth table of the constructed nodes. Therefore, when $F(s_1, \dots, s_{t-1}; f)$ holds and $F(s_1, \dots, s_{t-1}, s_t; f)$ needs to be evaluate, we simply traverse the generated circuit in a bottom-up manner, from the current wildcard node to be replaced to the root, to evaluate $g^{(t)(\boldsymbol{x})}$ for all $x \in \{0, 1\}^N$ with time complexity of $O(2^N \cdot d)$ in which d is the depth of the node. More details about the caching mechanism can be found in `check_conflict_batch_new` in `utils.py`.

Algorithm 2 Circuit generation with immediate equivalent node merging

Input: The Boolean function f that the generated circuit should be equivalent to. Next token prediction model $P(s_t|s_1,\ldots,s_{t-1})$.
Output: A feasible circuit g satisfying $g \in C(f)$.

Initialize $path$ as an empty stack of gates, POs as an empty list.
Initialize $G = \emptyset$ as a set of non-isomorphic gates
for $t = 1, 2, 3, \ldots$ **do**
  Compute a probability distribution of $s_t \in D$ by the next token prediction model

$$p_t \leftarrow P(s_t|s_1,\ldots,s_{t-1})$$

  Set $S_t \leftarrow \{s \in D | F(s_1,\ldots,s_{t-1},s;f) \text{ holds}\}$ ▷ $S_t \neq \emptyset$ is guaranteed by Characteristic 3.3
  $s_t \leftarrow \arg\max_{s \in S_t} p_t(s)$
  Initialize $s_t.input1 \leftarrow U, s_t.input2 \leftarrow U$ if s_t is a gate.
  if $path$ is empty **then** ▷ the output of s_t is the primary output of the circuit
    Append s_t to POs and push s_t to $path$
  else ▷ s_t should be the input of the last gate in $path$
    $s \leftarrow path.peek()$ ▷ get the last gate added to $path$
    if $s.input1 = U$ **then** $s.input1 \leftarrow s_t$ **else** $s.input2 \leftarrow s_t$ ▷ replace a wildcard node in s_t to s
    if s_t is a gate **then**
      $path.push(s_t)$
    else ▷ s_t is an input node in $x_1, \overline{x_1}, \ldots, x_N, \overline{x_N}$. Pop fully constructed gates from $path$
      while $s.input1 \neq U$ **and** $s.input2 \neq U$ **do**
        if $s \in G$ **then** ▷ Compute the truth table of s to check functional equivalence
          Update $path$ and POs to replace s with the functional equivalent one in G
        else
          Add s to G
        end if
        $s \leftarrow path.pop()$
      end while
    end if
  end if
end for
Return the circuit with POs as POs

Characteristic 3.3: Note that the AND gate $\wedge$ and the NAND gate $\overline{\wedge}$ is always in S_t in our sequential representation, as replacing a wildcard node to an AND or NAND gate with two wildcard nodes will never break the feasibility.

Characteristic 3.4: For all $g \in C(f)$, the sequence $s_1, \ldots, s_n$ that represents g is demonstrated in Section 3.4.

A.3 Demonstration of The Tree Positional Encoding

For example, in Figure 2, the position of the second node in the sequence, i.e., the uppermost AND gate connecting x_1 and x_2, can be represented as $e_2 = [10]$ (this gate's output is the first input of the rightmost AND gate, so "10" is pushed in the encoding stack of the rightmost AND gate, which is empty), and the position of the fourth node x_2 in the sequence can be represented as $e_4 = [01; e_2] = [0110]$ (push "01" in e_2 as x_2 is the second node's second input) and $e_6 = [10; e_5] = [1001]$ when x_2 is secondly visited as the fifth node's first input.

For circuits with multiple primary outputs ($M > 1$), we initialize the encoding stack of each primary output with a unique one-hot encoding, as if there is a virtual root node of M children, and each primary output corresponds to one of the children.

A.4 Immediate Equivalent Node Merging and Functional Equivalence Checking

With a depth-first replacement order, we can follow Algorithm 2 to merge equivalent nodes during the generation process. For functional equivalence checking of two nodes p and q, we check whether $p(\boldsymbol{x}) = q(\boldsymbol{x}), \forall \boldsymbol{x} \in \{0,1\}^N$ by iterating all $\boldsymbol{x}$. If there is an $\boldsymbol{x}$ such that $p(\boldsymbol{x}) \neq q(\boldsymbol{x})$, then p and q are not functionally equivalent.

Algorithm 3 Circuit Generation with Monte-Carlo Tree Search

Input: The Boolean function f that the generated circuit should be equivalent to. Next token prediction model $P(s_t|s_1, \ldots, s_{t-1})$. Immediate reward function $R(s_1, \ldots, s_t)$. Number of playouts K.
Output: A feasible circuit g satisfying $g \in C(f)$.
procedure PUCT(an MCTS node x)
  for a in x's all child nodes **do**
    $s_a \leftarrow \frac{a.\text{total\_value}}{a.\text{visited}} + a.\text{prob}\sqrt{\frac{x.\text{visited}}{1+a.\text{visited}}}$
  end for
  Return $\arg\max_a s_a$
end procedure
procedure MCTS(P, R, K) ▷ See [34] for details
  Create a root MCTS node r
  for $i = 1, 2, \ldots, K$ **do**
    (Selection) Starting from r, iteratively selects a child node via PUCT algorithm until reaching a leaf node l.
    (Expansion) Evaluate l via $P(s_t|s_1, \ldots, s_{t-1})$ (i.e., create all child nodes a for l, and assign a.prob for each child via $P(s_t|s_1, \ldots, s_{t-1})$), and select a valid child node (cutoff properties F holds) via PUCT.
    (Simulation) Run the generation process from the selected node via Algorithm 2 until a complete circuit is generated or the maximum #(iter) is reached, and get the cumulative reward v
    (Backpropagation) Update the "visited" and "total_value" attributes of the MCTS nodes from p to r in a backward pass with v.
  end for
  Return The path from r to the leaf node with maximal "total_value" attributes.
end procedure
Get optimized sequence $s_1, \ldots, s_t$ from MCTS(P, R, K).
Given $s_1, \ldots, s_t$, continue the generation process in Algorithm 2 and return the generated circuit.

Algorithm 4 Random generation of a k-input, l-output circuit

Input: Number of input k, number of output l, number of steps T.
Output: A randomly generated circuit with k inputs and l outputs.
$C \leftarrow [I_0, I_1, \ldots, I_{k-1}]$
for $i = 1, 2, \ldots, M_{\text{step}}$ **do**
  Create an AND node s_i
  Randomly sample two nodes $c_0, c_1 \in C$ without replacement
  Set the first input of s_i as c_0 or $\overline{c_0}$ randomly
  Set the second input of s_i as c_1 or $\overline{c_1}$ randomly
  Append s_i to the end of C
end for
Return the circuit with $I_0, I_1, \ldots, I_{k-1}$ as primary inputs and $a_{T-l+1}, a_{M-l+2}, \ldots, a_T$ as primary outputs.

A.5 Monte-Carlo Tree Search for Circuit Size Minimization

The search process is sketched in Algorithm 3.

A.6 Dataset Generation

The process to generate a random circuit is shown in Algorithm 4. We restrict that the length of the encoded sequence for each circuit should fit all the three sequential representations with a maximal length of 200, and all the 8 inputs should appear in the circuit. Each circuit has a unique structure, which is realized by a canonicalization technique [4].

A.7 Experiments Compute Resources

All the experiments are conducted on a workstation with the following specification:

- CPU: AMD Ryzen™ 9 7950X Desktop Processor (16 cores, 32 threads)
- Memory: 192GB (48GB $\times$ 4) DDR5 5200MHz
- GPU: NVIDIA GeForce RTX 4090 $\times$ 2

Each Transformer model in the experiments is trained on a single GPU with 75 hours.

NeurIPS Paper Checklist

1. **Claims**
   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   Answer: [Yes]
   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.
   Guidelines:
   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.
2. **Limitations**
   Question: Does the paper discuss the limitations of the work performed by the authors?
   Answer: [Yes]
   Justification: There are two main limitations for the proposed approach: (1) the sequential representation can be long when there are nodes with large fan-outs, discussed in Section 3.4; (2) it is still possible for Circuit Transformer to be unsuccessful in generating circuits due to the output length limit, which is discussed in section 4.
   Guidelines:
   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.
3. **Theory Assumptions and Proofs**
   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?
   Answer: [Yes]
   Justification: A proof is provided in Section A.2.
   Guidelines:
   - The answer NA means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Adequate implementation details and pseudocode are provided in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: See the supplemental material for the data and code.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not

including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**
Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?
Answer: [Yes]
Justification: See section 4 and Section A.6 for the details.
Guidelines:
    - The answer NA means that the paper does not include experiments.
    - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
    - The full details can be provided either with the code, in appendix, or as supplemental material.
7. **Experiment Statistical Significance**
Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?
Answer: [No]
Justification: The evaluation is conducted on two large datasets, each with more than 10 thousand circuits. The large size mitigates the statistical error, and repeated evaluation with re-sampling of datasets could be too time consuming.
Guidelines:
    - The answer NA means that the paper does not include experiments.
    - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
    - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
    - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
    - The assumptions made should be given (e.g., Normally distributed errors).
    - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
    - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
    - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
    - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.
8. **Experiments Compute Resources**
Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?
Answer: [Yes]

Justification: See Section A.7.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: No significant positive or negative societal impact is identified by the authors.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: No data or models that have a high risk for misuse are identified by the authors in this work.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The main third-party assets used in this paper, TensorFlow Models (Apache 2.0 License) and IWLS 2023 Benchmark (No license information) are cited in section 4.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**
    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?
    Answer: [NA]
    Justification: The paper does not involve crowdsourcing nor research with human subjects
    Guidelines:
    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.