Analog Integrated Circuit Topology Synthesis With Deep Reinforcement Learning

Zhenxin Zhao, *Member, IEEE*, and Lihong Zhang, *Member, IEEE*

***Abstract*—This article presents a novel deep-reinforcement-learning-based method for topology synthesis of analog-integrated circuits, especially operational amplifiers (OpAmps). It behaves like a human designer, who learns from trials, derives design knowledge and experience, and evolves gradually to finally figure out optimal manners to construct proper circuit topologies that meet design specifications. Essential design rules are defined and applied to set up the specialized environment for reinforcement learning in order to reasonably construct circuit topologies with building blocks as the basic components. Our proposed method can not only handle large-size circuit designs but also generate creative circuit topologies. The produced circuit topologies are verified by the simulation-in-loop sizing. In order to improve the evaluation efficiency, hash table and symbolic analysis techniques are utilized to significantly reduce the number of the produced topologies to be sized during the synthesis process. Compared with the state-of-the-art approaches, our proposed method significantly improves the synthesis efficiency by consuming only several hours on average to produce a trustworthy solution. Our experimental results demonstrate its sound efficiency, strong reliability, and wide applicability.**

***Index Terms*—Circuit topology synthesis, deep reinforcement learning (RL), parallel computing, symbolic analysis.**

I. Introduction

CONTINUOUS scaling of integrated circuit (IC) technologies and growing demands for high-performance low-power solutions have led to increasing difficulty in manual design of analog ICs [1]. Thus, efficient and automated analog circuit design is in great demand to facilitate laborious human effort, improve design productivity, and enhance design accuracy. The analog IC design flow contains four phases from high to low levels: 1) circuit synthesis; 2) physical layout design [2]; 3) parasitics extraction [3]; and 4) validation. An iteration may take place among those phases. Currently, almost all the low-level three phases are equipped with electronic design automation (EDA) commercial tools, even though they are still not as mature as their digital counterparts. However, automated analog circuit synthesis is still in its infancy stage with no commercial solutions available yet.

Following a typical industrial circuit synthesis flow, analog IC designers often manually select a circuit topology from a predefined library by considering tradeoffs among power, performance, area, yield, etc. However, the following sizing process, where EDA commercial tools have been already available to utilize, can only produce a result as good as the selected topology allows [4]. In order to meet the target specifications, other topology choices in the library may be also tried, which always leads to an expensive iteration. If necessary, a novel circuit topology has to be manually designed and verified. Therefore, this topology selection requires strong involvement from analog designers in terms of design knowledge and experience. But learning analog circuit design is a process that normally takes years to get started and decades to become a master. Thus, it is highly desirable for EDA tools to help designers in synthesizing circuit topology, which has motivated our work in this article.

However, one of the main challenges of commercializing analog circuit topology synthesis EDA tools is to solve the issue of the extremely low synthesis efficiency since the existing research works on this topic all suffer from unaffordable computation efforts, leading to a barrier to their industrial applications in practice. There are two effective approaches to reduce the computation effort. One is to speed up the topology evaluation (i.e., sizing) process. Another is to decrease the number of circuit topologies to be sized in the synthesis process. For the former approach, with the help of machine learning, accurate and efficient performance models [5]–[7] are proposed to substitute the time-consuming simulation program with IC emphasis (SPICE) simulator to reduce the sizing time, or some heuristic methods [8] and machine learning methods [9] are proposed to shrink the sizing iterations. For the latter one, performance boundary exploration [10] and fast evaluation [11] are widely used as filters to reduce the number of circuit topologies to be sized.

This article proposes a novel mechanism to reduce the number of circuit topologies to be sized by utilizing the merits of deep reinforcement learning (RL), which is able to significantly cut the computation effort of the synthesis process. To the best of our knowledge, this is the first work that employs the RL technique to automate the topology synthesis of analog ICs. It has the following remarkable features.

1) The synthesis process is a decision-making process.
2) Design search space is flexible and controllable.
3) Design knowledge is automatically learned.

Manuscript received 15 September 2021; revised 19 December 2021; accepted 20 January 2022. Date of publication 23 February 2022; date of current version 22 November 2022. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC); in part by the Canada Foundation for Innovation (CFI); in part by the Provincial Government of Newfoundland and Labrador; and in part by the Memorial University of Newfoundland. This article was recommended by Associate Editor S. Mohanty. *(Corresponding author: Zhenxin Zhao.)*

The authors are with the Department of Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL A1C 5S7, Canada (e-mail: zz4376@mun.ca; lzhang@mun.ca).

Digital Object Identifier 10.1109/TCAD.2022.3153437

4) Considerably less computation effort compared with the conventional topology generation methods.
5) Output of trustworthy solutions with detailed device sizes.

The remainder of this article is organized as follows. Section II reviews the previous relevant works. Section III explains the proposed deep-reinforcement-learning (DRL)-based circuit topology synthesis framework. Section IV elaborates on our defined specialized RL environment for circuit topology synthesis. The efficient performance evaluation of the produced topologies is illustrated in Section V. Section VI reports our experimental results and Section VII concludes this article.

II. Literature Review

To release the involvement from circuit designers, OASYS [12] manually creates templates of design styles in advance based on detailed design knowledge in order to facilitate its automated topology selection strategy. However, even though the selection process is done automatically, the time-consuming manual template design is unaffordable and hard to be generalized. AMGIE [13] solves the drawback of OASYS by automatically establishing a database at runtime through a large number of SPICE simulations, and then its proposed elimination strategy is applied to choose topologies based on the database. Unfortunately, this method only supports a few types of topologies, and building the database takes substantial computational effort. In general, those automated analog circuit synthesis methods in the category of topology selection have to address their time-consuming library setup challenge and poor capacity of generalization.

Due to its intractable shortcomings, the stream of the topology selection synthesis has phased out over the years. Instead, synthesis through topology generation, which can automatically generate circuit topologies from scratch, has attracted increasing research interests [14]. The existing automated circuit topology generation works can be mainly classified into two categories: 1) evolution algorithm (EA) based and 2) graph based. The EA-based circuit topology generation is typically realized through population-based metaheuristic optimization algorithms, such as the genetic algorithm and evolutionary algorithm, where circuit topologies are encoded in the forms of string [15], tree [16], graph [17], or matrix [18]. In the evolution process, the encoded circuit topologies in a generation are evaluated for their fitness. Then, new circuit topologies, which are bred through genetic (e.g., crossover or mutation) operations of the fittest topologies, will replace the least-fit ones to form a new generation. In this way, the synthesis process gradually evolves to eventually generate some satisfactory circuits that can meet the specifications. However, since the genetic operations are typically performed randomly without clear direction, the methods in this category often suffer from substantial computation effort wasted on generating and evaluating meaningless circuit topologies. Furthermore, there is no means to flexibly extend its search space once the rules of genetic operators have been designed, leading to a narrow range of circuits that can be synthesized.

The graph-based topology generation methods encode the topology generation process as a graph construction process in either a composition or decomposition manner. Each graph node represents a circuit building block (BB), which might be a resistor, a capacitor, a transistor, or any subcircuit, with input and output terminals. The composition process [19] or decomposition process [20], [21] is carried out by matching the input–output terminals of the graph nodes, which are guided by the defined composition rules or decomposition rules, respectively. Then, the constructed graphs are decoded into circuit topologies by mapping the graph nodes into their corresponding BBs as per a predefined BB library (PBBL). Compared with the EA-based generation methods, the design search space of the graph-based generation methods can be flexibly extended (even to infinitely large) depending on the allowed graph size (a user-defined parameter) and complexity of the PBBL. On the one hand, this feature is beneficial for generating a wider range of circuits, including some novel circuits. On the other hand, it would cause a severe problem, that is, topology explosion (i.e., a huge number of topologies generated in the synthesis process), because the graph-based approaches would explore all the possibilities in a brute-force manner within their allowed search space. Checking the feasibility of such a huge number of circuit topologies would be an extremely time-consuming computation task in practice.

Instead of generating circuit topologies from scratch as the topology generation methods, topology refinement, another branch of circuit topology synthesis methods, aims to modify an existing circuit topology with alternative BBs in the synthesis process to gradually improve performance to satisfy the target specifications [22]. In the topology refinement methods, a bunch of trustworthy circuits are analyzed with symbolic analysis [23] or SPICE simulation to extract the design knowledge, including performance capabilities, performance attribute tradeoffs, and parameter sensitivity [24], [25]. Such knowledge has multiple representations, such as topological similarity and causality information. The synthesis procedure acts as a decision-making process, where the starting subcircuit is reasoned, the distance to the required specification is calculated in each synthesis step, and the decision is made based on the knowledge toward the right direction. However, even though the methods in this category are able to produce certain sound solutions, the extremely time-consuming knowledge mining process, which may require considerable involvement from human experts, makes it very costly to be generalized for synthesizing another class of circuits.

To address the shortcomings of the existing topology synthesis approaches, in this article, we propose a novel DRL-based circuit topology synthesis methodology. DRL is a machine learning technique known to be able to solve complex tasks. It has been successfully applied in some EDA areas to facilitate circuit design, such as analog circuit sizing [26]–[28] and analog layout placement [29]. Our proposed DRL-based method not only integrates the merits together of both topology generation methods and topology refinement methods but also overcomes their drawbacks. Specifically, the proposed DRL-based method starts with a starting subcircuit and gradually expands it with BBs through selecting and executing actions, which is similar to the topology refinement methods, but the design knowledge used for making decision in our method is automatically learned without any intervention from humans. Besides using BBs as the basic components, the composition and decomposition rules employed in the graph-based topology generation methods are effectively defined in our proposed DRL-based method as rule-based actions, which function in a similar way to reasonably construct circuit topologies in the synthesis process, but the synthesis efficiency of our proposed method is significantly higher than the conventional topology generation methods.

The performance of a circuit not only depends on its topology (i.e., circuit structure) but also relies on its device sizes. In our proposed method, we justify the feasibility of a produced circuit topology by checking whether it is able to meet a certain performance specification through a simulation-in-loop sizing operation rather than seeking for an optimum performance. As long as the performance of a produced topology meets the specification requirement, the DRL agent will receive a positive reward; otherwise, it will get a negative reward. Through successive trials, the DRL agent is continuously learning from the rewards and improving its means to reach a solution (i.e., a feasible topology) by figuring out which BBs to expand is the best choice at each construction step. Moreover, hash table and symbolic analysis are employed in our work to reduce the number of produced circuit topologies to be sized during the synthesis process, which would significantly boost the whole synthesis efficiency. In addition, the parallel computing technique is adopted to speed up the time-consuming circuit sizing process.

III. Proposed Framework

In this section, we will first introduce the basic concepts of DRL, then discuss its detailed application into our proposed circuit topology synthesis framework, and finally present our proposed circuit topology synthesis system.

A. Preliminary

RL is built upon an agent that iterates to function in an environment using a trial-and-error process that mimics learning in humans, where agent is defined as an imagined learner (i.e., the algorithm itself) and environment is deemed as the world where the agent operates [30]. There are three basic elements of RL, which are *state*, *action*, and *reward*. All of them are associated with the RL environment, which is based on the problem to be solved. Specifically, *state* is the observation of the status for the environment, *action* is the operation that the agent performs in the environment, and *reward* is the feedback that the agent receives from the environment after making an action.

DRL combines RL and deep learning (DL), as reflected by the fact that the RL agent is connected with a neural network (NN). At each time step, the RL agent observes the current state of the environment, consults the NN by inputting the current state, infers the action to be taken based on the output of NN, and takes the selected action. The environment then returns a new state that is used to calculate the reward

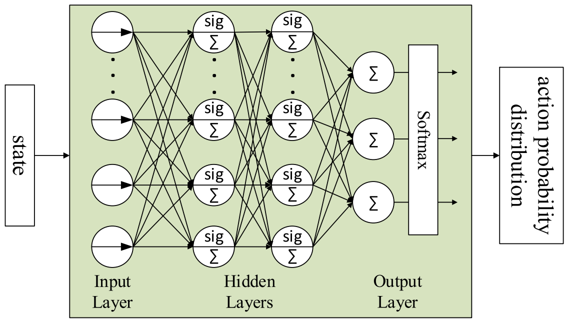


Fig. 1. Policy gradient neural network (PGNN).

as a result of taking that particular action. In the meantime, the current state, selected action, and resultant reward are collected as training data to train the NN through DL, which will help the RL agent make better decisions subsequently. This continuous RL loop would eventually help the agent to learn to act in a way (i.e., finding a sequence of actions) that will maximize its expected cumulative reward, which is the objective of RL [30]. The *discounted future reward* G_t at time step t can be written as

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

where r_{t+k+1} is the reward received at time step $t + k + 1$ (k could be any value between 0 and infinity since the termination time step of this RL loop is uncertain); and $\gamma \in (0, 1]$ is called the *discount factor*, which means future rewards are worthless than immediate rewards due to the common sense that rewards coming sooner are more likely to take place.

There are several schemes that have been proposed to carry out the DRL process, such as Q-learning, policy gradient, and advantage actor–critic [31]. The circuit topology synthesis process, which we have to deal with in this work, features the following special situation and in turn, the uncommon DRL environment: any circuit topology can only be evaluated after a valid structure, which satisfies the circuit input-output specification, has been constructed. Compared with either Q-learning or advantage actor–critic method, the policy gradient method shows its best fit to our work because this method can wait to calculate the reward until the end of an *episode* (or called *trajectory*), which is defined as a sequence of states, actions, and rewards from an initial state to a termination state.

The policy-gradient-based method attempts to directly find a policy that can form a circuit topology meeting the target specifications. Here, *policy* is defined as the strategy that an agent employs to determine the next action based on the current state [30]. In the policy-gradient-based DRL, the policy (π) is modeled as a policy gradient NN (PGNN) with parameterized weights θ. As depicted in Fig. 1, the PGNN is actually a supervised classification model, which takes a state (s) as input and outputs the probability distribution over all actions (a)

$$\pi_\theta(a|s) = P(a|s). \tag{2}$$

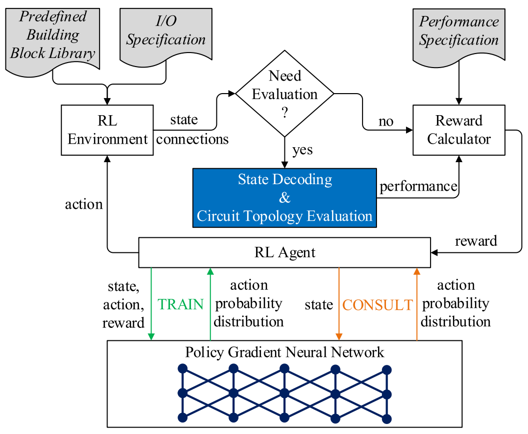

Predefined Building Block Library
I/O Specification
Performance Specification
Need Evaluation ?
RL Environment
state connections
no
Reward Calculator
yes
State Decoding & Circuit Topology Evaluation
performance
action
reward
RL Agent
state, action, reward
TRAIN
action probability distribution
state
CONSULT
action probability distribution
Policy Gradient Neural Network

Fig. 2. Proposed DRL framework for circuit topology synthesis.

The architecture of PGNN is a fully connected artificial NN (ANN), which is comprised of a passthrough input layer, several hidden layers, and an output layer. The activation functions applied in the hidden and output layers are the widely used sigmoid function and softmax function, respectively.

B. DRL Framework for Circuit Topology Synthesis

As depicted in Fig. 2, in our work, the automated circuit topology synthesis is realized through the proposed DRL framework. The terms (e.g., state, action, and reward) used in Fig. 2 are the basic elements of RL as presented in Section III-A. BBs are utilized as the basic components, which are selected from a PBBL by taking actions, to construct circuit topologies. The design specifications are composed of input–output (I/O) specification and performance specification. Among them, the I/O specification works with the PBBL to form the specialized RL environment for circuit topology synthesis, while the performance specification is utilized to calculate the reward for the RL agent. It is worth noticing that only the states that satisfy certain conditions, which will be explained in more detail in Section V-B, need to be decoded into circuit topologies for evaluation. For the states that do not meet the conditions, a reward will be directly assigned according to our reward function as defined in Section IV-D.

Unlike the consulting of PGNN that occurs at all the time steps, the training of PGNN is carried out only at the end of an episode. Thereby, the training data contain the collected states, actions, and rewards of one or multiple episodes. Let π_θ denote a policy π with parameters θ, and $J(\theta)$ denote the expected ($\mathbb{E}$) cumulative discounted reward of an episode, which is also called the objective function. Then, the gradient of $J(\theta)$ can be expressed by the following [30]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left(\sum_{t=0}^{T} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right) \tag{3}$$

where t is a time step within the episode and T is the total time step of the episode. The gradient ascent method is employed to iteratively find the best weights that improve the policy in the direction of maximizing $J(\theta)$ with the following update rule:

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta) \tag{4}$$

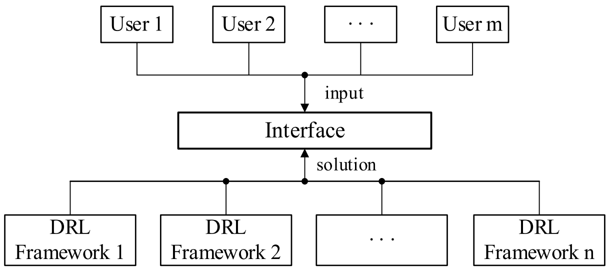


Fig. 3. Proposed circuit synthesis system.

where α is the learning rate. Through episodic training, the agent gradually learns the essential synthesis knowledge, which is stored in the PGNN. This training process terminates when the gradient [i.e., (3)] equals zero, which means that the weights of PGNN are not able to be further optimized.

As illustrated in Fig. 2, the consulting and training of PGNN are carried out simultaneously during the whole learning process. Specifically, the consulting of PGNN will get feedback from the environment via rewards, which would be used to train the PGNN in return to let it learn from the trial results and be able to make better decisions subsequently based on the learned knowledge. After the PGNN has been well trained, the RL agent can always derive a solution in its first episode trial.

C. Circuit Topology Synthesis System

As explained above, the proposed DRL framework is able to return a synthesized circuit solution after conducting a learning process. The knowledge of the way to construct a feasible circuit structure from the starting subcircuit as encoded in its initial state is stored in the PGNN. Therefore, with various initial states preferred by different human circuit designers, the PGNN would possess the knowledge of the ways to construct circuit structures from distinct starting subcircuits after good training. Furthermore, users may also have various I/O and performance specifications for circuit design, which needs distinct RL environments to address. As shown in Fig. 3, the proposed circuit synthesis system contains multiple DRL frameworks, each of which possesses a unique RL environment to deal with a target design specification. Therefore, this system is able to provide trustworthy solutions with detailed device sizes to the users as long as their input starting subcircuits and design specifications are reasonable. The more users there are, the smarter the DRL circuit synthesis system becomes. If the input (i.e., starting subcircuit and design speicification) has already been learned, this system will return a valid solution right away. Otherwise, this system has to go through a learning process and then return a solution as output.

IV. Specialized RL Environment

This section will explain all the components contained in our proposed specialized RL environment in detail for analog circuit topology synthesis, including BB, state, action, episode, and reward.

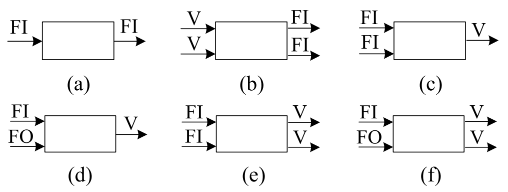


Fig. 4. Examples of the six types of BBs. (a) One-signal-path BB. (b) Two-signal-paths BB. (c) Identical-current converter with one output. (d) Distinct-current converter with one output. (e) Identical-current converter with two outputs. (f) Distinct-current converter with two outputs.

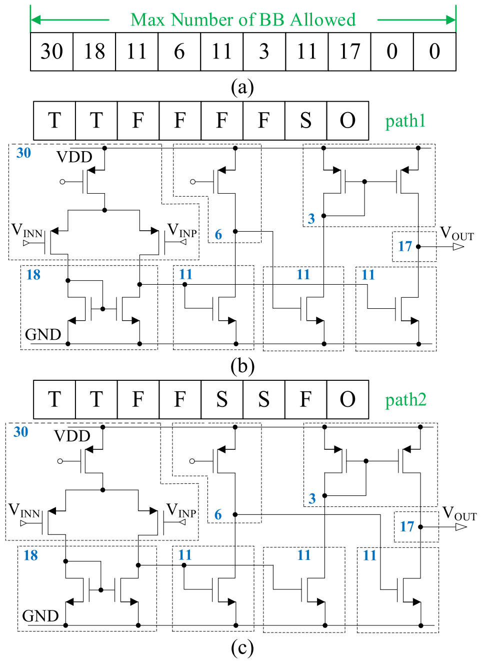


Fig. 5. Example state and its possible represented circuit topologies. (a) State A. (b) Circuit topology represented by the combination of state A and path1. (c) Circuit topology represented by the combination of state A and path2.

A. Building Block

As mentioned in Sections II and III, our proposed work utilizes BBs as the basic components to construct circuit topologies. The input and output terminals of a BB can be categorized into three types: *V*, FI, and FO. Among them, *V* means the corresponding input or output terminal is a voltage terminal, while FI or FO represents a current terminal with bias current flows into or out of a BB, respectively. Based on the counts and types of input and output terminals, we have defined six types of BBs. The names of these types are self-explained. Fig. 4 illustrates an example of each type of BB, where the input and output terminals always lie on the left-hand side and right-hand side of a block, respectively. Generally, each input or output terminal can be either *V*, FI, or FO. But for all the converter-type BBs, the input terminals must be current while the output terminal(s) must be voltage. In addition, the two-signal-paths BBs must have the same type of input terminals as well as the same type of output terminals. Some example BBs are depicted in Table I in Section VI, which are used for conducting our experiments in this work.

B. State

In our proposed DRL-based framework, states are represented by the data structure of array. The value at each slot in the array refers to the index of a BB in the PBBL if it is not equal to zero. Otherwise, it means no BB (i.e., empty) is available in this slot. In this way, each state encodes a sequence of BBs, making up a circuit topology. An example state (state A) is given in Fig. 5(a). The size of a state is determined by the maximum number of BB allowed (a user-defined parameter) for constructing a circuit topology. This constraint is necessary to practically limit the design search space. According to our experiments, this useful constraint works well with the extendable PBBL, contributing to flexible and controllable design search space in our proposed circuit topology synthesis methodology.

However, if the detailed connections among the represented BBs are unclear, one state may be able to be decoded into several circuit topologies. To address this issue, we define an attribute called *path*, which is also an array, to track the signal path that each BB in the corresponding slot belongs to. There are four types of *path* in total: *F*, *S*, *T*, and *O*, which means that the added BB belongs to the first signal path, the second signal path, both signal paths, and the single signal path, respectively. If adding a BB leads to the signal path of the current structure changing from nonsingle to single, this BB should also be tracked as *O*-type *path*. During the synthesis process, the path information is recorded in order to assist the decoding of a state because the combination of a pair of state and path would lead to a determined circuit topology correspondence. Fig. 5(b) and (c) illustrates two possible paths and their represented circuit topologies when being combined with state A.

C. Action

In our proposed framework, taking an action means selecting a BB from the PBBL to connect with the output terminal(s) of the current-state-represented circuit structure. Although the DRL has the ability to automatically learn the optimal selection strategy through its continuous trial-and-error process, making the DRL process follow some basic design rules would largely avoid constructing meaningless circuit structures, contributing to significant speed-up of the learning process. These basic design rules force the connected terminals to be matched. Specifically, *V* must be matched with *V*, FI with FO, and vice versa. Based on these rules, the actions can be divided into three types: 1) unmatched action; 2) matched action; and 3) terminational action. In this regard, if any input terminal of the BB selected by an action cannot match with the output terminal that it is going to connect, this action is called an unmatched action. Otherwise, it is named as a matched action. The terminational action is a special type of action, which is only allowed to happen when the output terminal(s) of the current structure has (have) the same counts and types as the output specification required. Executing a terminational action would not select any BB to connect with the current structure, but cause the termination of an episode.

Based on the types of BBs selected by actions, the matched actions can be categorized into single match, symmetric match,

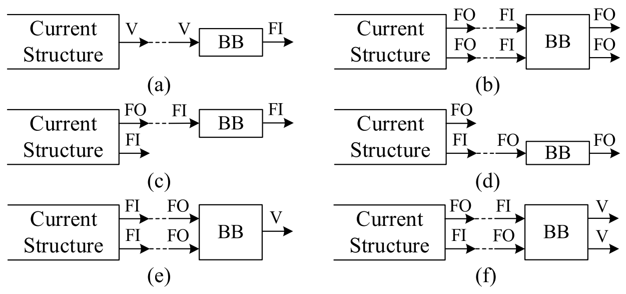


Fig. 6. Examples of the matched actions. (a) Single match. (b) Symmetric match/normal match case1. (c) Normal match case2. (d) Normal match case3. (e) Normal match case4. (f) Normal match case5.

and normal match. Besides the *path* information, knowing the type of a match would greatly facilitate the process of figuring out the detailed connections between the current structure and the added BB. An example of each type of matched action is illustrated in Fig. 6. Their formal descriptions are listed as follows.

1) *Single Match:* Connect a one-signal-path BB to a one-output structure.
2) *Symmetric Match:* Connect a two-signal-paths BB to a two-identical-outputs structure that is symmetric.
3) *Normal Match Case1:* Connect a two-signal-paths BB to a two-identical-outputs structure that is asymmetric.
4) *Normal Match Case2:* Connect a one-signal-path BB to the first output terminal of a two-outputs structure.
5) *Normal Match Case3:* Connect a one-signal-path BB to the second output terminal of a two-outputs structure.
6) *Normal Match Case4:* Connect an identical-current converter to a two-identical-outputs structure.
7) *Normal Match Case5:* Connect a distinct-current converter to a two-distinct-outputs structure.

In order to further reduce the chance of generating senseless circuit topologies, it is essential to respect necessary structural symmetry constraints in circuit design, such as the first stage of operational amplifiers (OpAmps) (due to the preference of differential pairs) [32]. In the proposed work, we keep tracking the symmetry property of the structure being constructed. At the beginning of an episode, symmetric-match actions are allowed. However, once a normal-match action is taken, which would break the symmetry property of the structure being constructed, the symmetry-match actions are no longer allowed to perform within this episode. Thereby, although the normal match case1 shares the same connection way as the symmetric match, they are different since the normal match case1 is still allowed to take when the symmetry property has already been broken.

D. Episode and Reward

As mentioned in Section III, the policy-gradient-based RL is carried out episodically. Within one learning task, all the episodes start with the same initial state. This initial state could be an encoded BB or a subcircuit as long as its input terminal (or terminals) meets (or meet) the input specification. At each time step, the RL agent selects an action to take. After executing the action, the state transforms into a new state, which

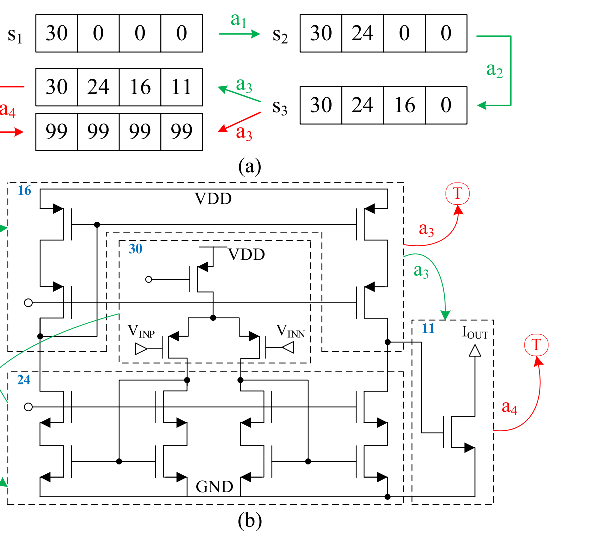


Fig. 7. Two example episodes. (a) State transitions. (b) Corresponding circuit topology construction processes.

has one more slot becoming nonzero. This process continues until entering a termination state, which indicates the ending of an episode. In our work, we have defined a uniform format for the termination state, in which all the slots have the same value of 99. An episode would go to the termination state when any of the following three cases has occurred: 1) an unmatched action is taken; 2) state boundary is exceeded; 3) a terminational action is taken.

Fig. 7(a) and (b) depicts two example episodes and their encoded circuit topology construction processes, respectively. The two episodes both start with the 30th BB in PBBL. After taking actions a_1 and a_2, the 24th and 16th BBs in the PBBL are added in sequence, as shown by one more slot becoming nonzero in state at each time step. Now, the output of the structure being constructed becomes 1 V. Assuming it meets the output specification, then the terminational-type actions are allowed to take at this point. If an unmatched action or a terminational action [e.g., action a_3 in red on the lower branch in Fig. 7(a)] is taken, the state turns into the termination state and this episode is completed. Otherwise, a matched action (e.g., action a_3 in green on the upper branch) is made and the episode continues. After taking this matched action, the 11th BB is added into the state, which reaches the state boundary (i.e., all the slots in the state are filled with nonzeros). Then, taking any type of further action (e.g., action a_4 in purple) would let the episode end with the termination state due to the exceeding of the state boundary.

Assume t is the time step within an episode, and this episode terminates at the *T1*, *T2*, or *T3* step due to occurrence of case 1), 2), or 3), respectively. The reward function $r(t)$ is accordingly defined as follows:

$$r(t) = \begin{cases} 0, & \text{if } t < \min(T1, T2, T3) \\ -5, & \text{elif } t = T1 \\ -3, & \text{elif } t = T2 \\ \left.\begin{matrix} 1, & \text{if meeting spec.} \\ -1, & \text{otherwise} \end{matrix}\right\} & \text{elif } t = T3. \end{cases} \quad (5)$$

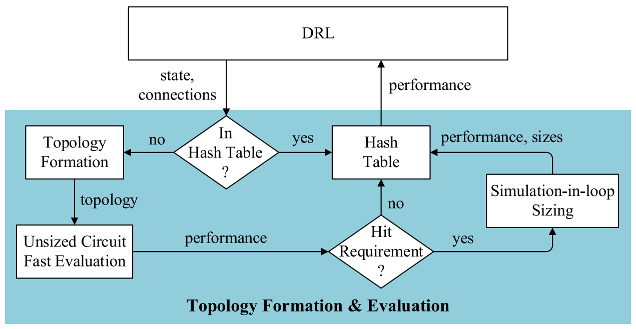


Fig. 8. Proposed framework with detailed circuit topology evaluation process.

As the reward function indicates, all the nontermination steps would receive a reward of 0, while for the termination step, a positive or negative reward is assigned to encourage or discourage the generation of the corresponding circuit topology, respectively. For the termination that occurs at *T3*, the constructed circuit topology has to be evaluated. If its performance meets the target specification, +1 is assigned. Otherwise, −1 is assigned.

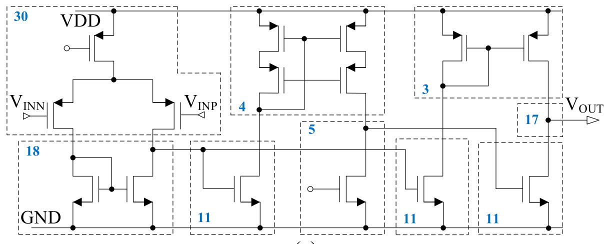
However, like the example shown in Fig. 7, even though taking actions a_1 and a_2 is in the correct direction to form a circuit topology that meets the target specifications, if action a_3 is an unmatched action, −1 will be received to make the agent feel that all the selected actions in this episode are generally in the wrong direction. Learning exactly which action is good or bad is quite slow, which may require a huge number of iterations to figure out. In order to speed up such a learning process, we need to tell the agent that actions a_1 and a_2 are actually good. Therefore, when a termination occurs at *T3* and the evaluated performance meets the target specification, we go back to modify the reward received from taking the second last action (e.g., a_2 in Fig. 7 for the case with a_3 in red leading to the termination state) to be +5 after completing this episode.

V. Circuit Topology Formation and Evaluation

During the RL process, when the produced states need to be evaluated, the simulation-in-loop sizing is applied in our work to verify their decoded circuit topologies. This is a time-consuming operation per se. In order to boost the evaluation efficiency, the hash table, symbolic analysis, and parallel computing techniques are employed in our work. The evaluation process is illustrated in Fig. 8, which is the detailed implementation of the shaded block in Fig. 2. Specifically, the state to be evaluated and its associated connections are fed to the hash table first to check its existence. If they already exist, their performance attributes are fetched and returned to the reward calculator. Otherwise, they need to be first decoded into a circuit topology and then evaluated by the proposed fast evaluation filter. Only the survived ones will go to the subsequent sizing phase to check the performance feasibility. Finally, the evaluated results of both fast evaluation and sizing are stored in the hash table.

A. Hash Table

The hash table guarantees that one circuit topology only needs to be evaluated once. In this way, the total evaluation time for the whole learning process is significantly reduced. However, one circuit topology may be represented by many combinations of state and connections if both normal-match-case2 and normal-match-case3 have taken place during the construction process. For instance, the circuit topology depicted in Fig. 9(a) can be represented by either the combination of state A and path1 or the combination of state B and path2 illustrated in Fig. 9(b). The difference between them is the occurrence order of the normal-match-case2 and normal-match-case3, which indicates that the BBs are added to the first signal path (*F*-path) and the second signal path (*S*-path) at distinct time steps. The following lemma is to answer whether their occurrence order will affect the produced circuit structure or not.



(a)

path	T	T	F	S	F	S	F	F	O	
state A	30	18	11	11	4	3	5	11	17	0
path	T	T	S	F	F	F	S	F	O	
state B	30	18	11	11	4	5	3	11	17	0

(b)

path	T	T	F	F	F	F	O	S	S	
ustate	30	18	11	4	5	11	17	11	3	0

(c)

Fig. 9. Example of unique state representations. (a) Circuit topology. (b) Two combinations of state and path. (c) Unique state representation of states A and B.

Lemma: If both normal-match-case2 and normal-match-case3 take place within a circuit construction process, the occurrence order between these two types of matches will not affect the constructed circuit structure as long as the orders among each type of match are maintained.

Proof:

1) It is a fact that at each time step, the order of adding an *F*-path-type or *S*-path-type BB to the current state will not affect the produced circuit structure if both of them have to take place. Thus, it can be inferred that adding an *F*-path-type (or *S*-path-type) BB can be postponed after adding one or multiple *S*-path-type (or *F*-path-type) BBs without affecting the produced circuit structure, as long as it is allowed at that time step. For example, in Fig. 9(b), the two *S*-path-type BBs are located at different slots in state A and state B but still represent the same circuit structure depicted in Fig. 9(a).
2) If the occurrence order of the same path-type BBs is not preserved, the input and output terminals of the connected BBs may no longer be matched, leading to an invalid circuit structure. For instance, exchanging the two values in the fourth and sixth slots of state A (i.e., both 3) in Fig. 9(b) will not alter it but cause unmatched

terminals among connected BBs, resulting in an invalid circuit structure. Based on the two analyses above, the lemma can be proved. ■

In order to save the size of the hash table, we need to convert the states due to those combinations, which are actually mapped to the same circuit topology, to a unique representation. That is to say, we need to set up a one-to-one correspondence between state representation and circuit topology. Here, we will first define an operation called *state equivalent moving operation:* in a state, moving all the S-path-type BBs to the position after the last nonzero BB while still preserving their original relative occurrence order.

Theorem: The state equivalent moving operation is able to convert any states, which represent the same circuit structure, to a unique state representation.

Proof: The multiple-to-1 correspondence between state representation and circuit topology is actually caused by the confusion due to multiple signal paths existing in the circuit topology construction. All the combinations that satisfy the Lemma described above can be viewed as equivalency in terms of the represented circuit structure. Thus, after such a moving operation, all the equivalent state representations should be converted into the same one, which is unique from others because the confusion source has been eliminated. Thus, the theorem must hold. ■

For example, after the state equivalent moving operation, state A and state B are converted into the same representation depicted in Fig. 9(c) (marked as *ustate*), which is unique. After converting a state to-be-evaluated to its unique representation *ustate*, its hash can be calculated via the following formula:

$$h(\text{ustate}) = \prod_{i=0}^{n} \text{prime}[j] \bmod \left(2^{32} - 5\right) \tag{6}$$

where i is the index of a slot in *ustate*, n is the number of nonzero slots in *ustate*, j is the slot value (i.e., the index of the corresponding BB) in *ustate*, *prime* is an array that stores all the prime numbers starting from 1 in the increasing order with the size being equal to the number of BBs in PBBL, and $2^{32} - 5$ is known to be the largest 32-bit unsigned prime number. Each circuit topology is put into a bucket according to its hash. For the circuit topologies that have only been fast evaluated, their dc gain results are stored in the hash table. For the ones that have been sized, their performance results and sizes are stored in the hash table.

B. State Decoding

As mentioned in Section III-B, only when a terminational action is taken, the second last state [e.g., s_3 on the lower branch in Fig. 7(a)] rather than the termination state needs to be decoded and evaluated. The states that satisfy this condition only occupy a very small portion of the total states generated in the synthesis process, which is one of the main reasons that our proposed DRL-based topology synthesis method features good efficiency.

The state to be decoded is first mapped to its represented BBs. Then, transistor-level circuit topologies will be formed by connecting those BBs according to the recorded connection information. In practice, simply swapping input pins of an analog circuit may affect its performance. Due to this fact, we treat the circuits, which have exactly the same structure but swapped input pins, as different circuit topologies. Therefore, the decoded circuit topology should create a copy of itself with swapped input pins.

Algorithm 1 DRL-Based Circuit Topology Synthesis

Input: I/O specification S_{IO}; Performance specification S_P; Initial state s_0; Batch size N; *PBBL*; Hash table T.

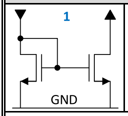
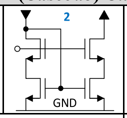
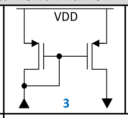
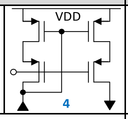
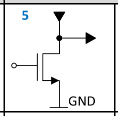
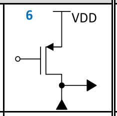
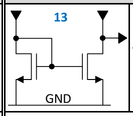
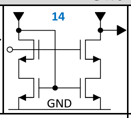
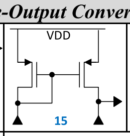
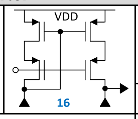
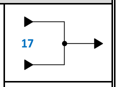
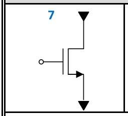
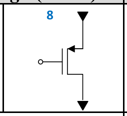
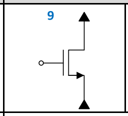
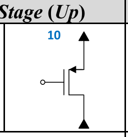
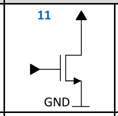
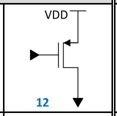
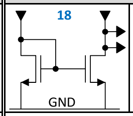
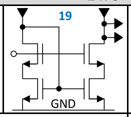
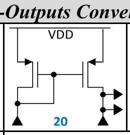
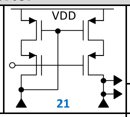
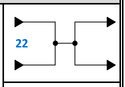
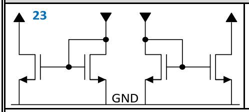
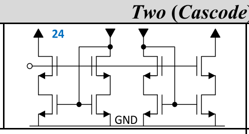
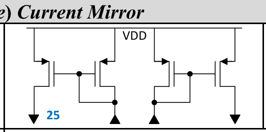
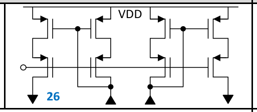
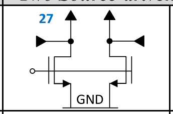
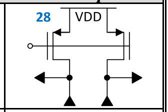
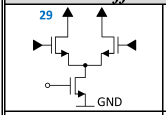
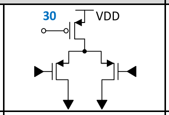
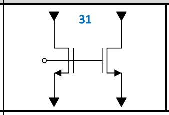
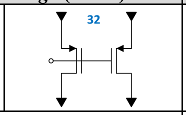
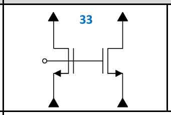
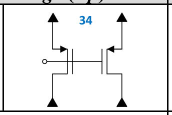
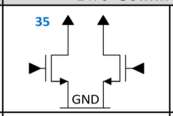
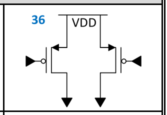
Output: Circuit with device sizes and corresponding performance.

```
1:  Build the RL environment based on $S_{IO}$ and $PBBL$;
2:  Build the PGNN, gradient function $\nabla_\theta$, and hash table $T$;
3:  **While** (true):
4:     Create empty lists $S$, $A$, $R$, $R_B$, $G_t$;
5:     **While** (true) {
6:        Start an episode with $s_0$:
7:        **If** (episode terminates):
8:           Calculate the discounted reward ($G_t$) of $R$;
9:           Store $G_t$ into set $R_B$;
10:          **If** (the size of $R_B >= N$): **break**;
11:          Clear list $R$ to be empty;
12:       Choose an action $a$ by using PGNN; Perform $a$;
13:       Receive reward $r$ according to $S_P$ (may insert a bucket
          into $T$); Get new state $s$;
14:       Store $s$, $a$, and $r$ into lists $S$, $A$, and $R$, respectively;}
15:    Train the PGNN with $S$, $A$, and $R_B$; Calculate $\nabla_\theta$;
16:    **If** ($\nabla_\theta$ == 0): **break**;
17: **End While**;
18: Starting an episode with $s_0$, get last state $s_f$ before
    termination;
19: Decode $s_f$ into circuit netlist $C$;
20: Extract the size $S$ and performance $P$ of $s_f$ from $T$;
21: **Return** $C$, $S$, and $P$;
```

C. Unsized-Circuit Fast Evaluation

As shown in Fig. 8, the decoded circuit topologies will first be fast evaluated, which can roughly assess quality of the topologies through symbolic analysis. In our proposed work, the graph-pair decision diagram (GPDD) algorithm is employed to numerically calculate the dc gain of an unsized circuit, which requires the values of all the parameters in the small-signal model of the circuit as input [21]. However, since the circuit to be evaluated in our context is unsized, it is impossible to get those values via SPICE simulation. What we know so far is that there exist value ranges for those small-signal model parameters in a certain technology. Therefore, after extracting such ranges for a specific technology, the center values of these ranges are employed to efficiently estimate the DC gain through the GPDD algorithm. Furthermore, in order to further speed up the operation, we employ a simplified small-signal model that only contains four parameters (g_m, g_{ds}, C_{gs}, and C_{gd}) for each metal-oxide-semiconductor field-effect transistor (MOSFET) in a circuit [33]. Due to

TABLE I
PREDEFINED BUILDING BLOCK LIBRARY (PBBL)

One-Signal-Path Building Blocks						Converter Building Blocks				
(Cascode) Current Mirror				*Current Source*		*One-Output Converter*				
										 FI|FO - V
FI - FI		FO - FO		FI - V	FO - V	FI|FI - V		FO|FO - V		FO|FI - V
Cascode Stage (Down)		*Cascode Stage (Up)*		*Common Source*		*Two-Outputs Converter*				
										 FI|FO - V|V
FI - FO		FO - FI		V - FI	V - FO	FI|FI - V|V		FO|FO - V|V		FO|FI - V|V
Two-Signal-Paths Building Blocks										
Two (Cascode) Current Mirror				*Two Source-driven Current Splitter*						
										
FI|FI - FI|FI		FO|FO - FO|FO		FI|FI - FI|FI	FO|FO - FO|FO					
Differential Pair		*Two Cascode Stage (Down)*		*Two Cascode Stage (Up)*		*Two Common Source*				
										
V|V - FI|FI	V|V - FO|FO	FI|FI - FO|FO		FO|FO - FI|FI		V|V - FI|FI	V|V - FO|FO			

approximate calculation, we deliberately lower the requirement of this quality filter compared to the given performance specification, which ensures that only the circuit topologies with very bad performance will not proceed to the subsequent sizing phase. Our experiment results in Section VI will show high effectiveness of this proposed fast evaluation filter.

D. Simulation-in-Loop Sizing

Once the circuit topologies pass the fast evaluation filter, they have to be sized to check their performance feasibility [34]. In this work, we employ the well-known nondominated sorting genetic algorithm II (NSGA-II) to size circuits, which utilizes the SPICE simulation to evaluate circuit performance. Since our purpose of sizing is to check the performance feasibility instead of optimizing the circuit, we have slightly modified the NSGA-II algorithm. Specifically, the sizes of population and generation are set to 20 and 50, respectively. During the algorithm running, if the evaluated performance of any individual meets the target performance specification, the algorithm terminates right away and returns the performances and device sizes to the hash table. Otherwise, after executing all 50 generations, the best performances achieved so far and the corresponding device sizes are stored in the hash table. At each time, there are two circuit structures with swapped input pins to be sized. To boost the sizing efficiency, we employ the parallel computation technique to size the circuits.

The whole DRL-based circuit topology synthesis process is illustrated in Algorithm 1. Specifically, lines 1 and 2 set up the specialized RL environment for circuit topology synthesis and the PGNN for learning synthesis knowledge, lines 4–14 collect the training data by interacting with the environment, and line 15 utilizes the training data to train the PGNN. The collecting and training processes are repeated until the gradient function equals zero. After that, lines 18–20 extract the solution, which includes circuit netlist, performance results, and corresponding sizes, from the trained DRL-based framework.

VI. EXPERIMENTAL RESULTS

The proposed circuit topology synthesis framework was mostly implemented in Python, with the fast evaluation (GPDD algorithm) realized in C++, the NSGA-II sizing carried out in C, and the SPICE simulations conducted by the Cadence tool. Our experiments were run on an Intel X86 1.2-GHz Linux workstation that has 64 GB of memory. All the experiments were conducted in a CMOS 65-nm technology process, which can be readily replaced by any other CMOS technologies.

A. Experimental Parameter Settings

As shown in Table I, our defined PBBL contains 36 BBs, each of which has a unique index. All our experiments were carried out based on this PBBL, which includes 12 one-signal-path BBs, 14 two-signal-path BBs, and 10 converter-type BBs. The one-signal-path BBs are composed of well-known current mirrors, Cascode current mirrors, current sources, Cascode stages, and common sources. Besides the differential pairs, the two-signal-paths BBs are basically made up of two copies of one-signal-path BBs. Among the converter-type BBs, there are two special BBs that only contain nets, while the others are the variants of the (Cascode) current mirrors.

Each type of BB could be composed of N-type MOSFET (nMOS) or P-type MOSFET (pMOS) transistors, leading to distinct terminal types. For different types of BBs, their input and output terminals are expressed in distinct formats as follows: 1) One-signal-path type BBs: *input–output*;

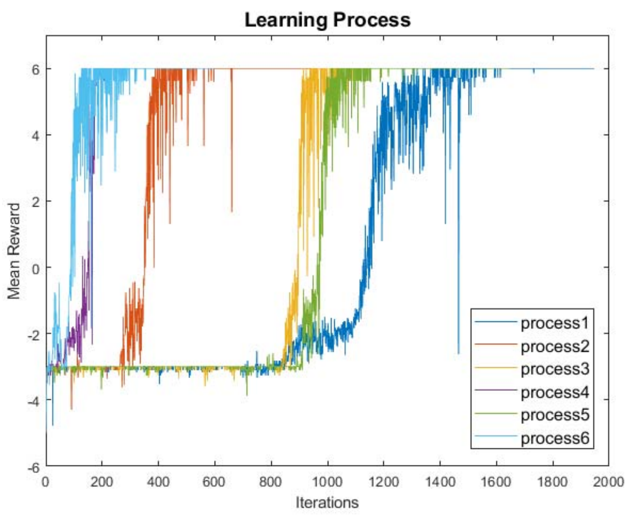


Fig. 10. Mean reward of a batch during the learning process.

2) Two-signal-paths type BBs: *input1*|*input2*–*output1*|*output2*; 3) Converter type BBs: *input1*|*input2*–*output* or *input1*|*input2*–*output1*|*output2*. It is worth mentioning that the two special nets-contained-only BBs (i.e., the 17th and 22nd BB) both have two possible combinations of input-output terminals, which are completely dependent on the order of the input terminals. In addition, in order to facilitate the job of evaluating the produced circuit topologies, we require that except for the differential pairs (i.e., the 29th and 30th BBs in the PBBL), all the transistors within a BB have the same length and the same width.

In our experiments, the size of a state was set to be 10, which is the maximum number of BBs allowed to construct circuit topologies. The size of action was set to be 20, which equals the maximum number of possible choices for expanding a BB. For the PGNN, we defined two hidden layers, with the first and second layers containing 300 and 200 neurons, respectively. The learning rate, discount factor, and batch size were set to be 0.002, 0.95, and 200, respectively.

B. Analysis of the Learning Process

To synthesize OpAmp circuits, we set the starting subcircuit as a differential pair made of pMOS (i.e., the 30th BB), the input–output specification as two voltage inputs and one voltage output, and the performance specification as 60-dB dc gain (*Av*), 60° phase margin (PM), 10-dB gain margin (GM), and 10-MHz unity-gain bandwidth (UGB). In order to fairly illustrate the learning process of the RL agent in the proposed framework of circuit topology synthesis, we ran the algorithm six times and received six corresponding learning processes, which are depicted in Fig. 10.

As shown in the diagram, these six processes terminate at different iterations. Specifically, process2, process4, and process6 finished within 800 iterations, while process1, process3, and process5 required around 2000, 1300, and 1600 iterations, respectively. However, all these six learning processes share almost the same learning trend. Specifically, in the beginning, all the processes got the mean reward around −5, which means that their episodes within a batch were mainly terminated due to unmatched actions taken. Then, the mean of the received rewards in a batch became −3. It indicates that the RL agents figured out taking unmatched actions was bad and tried to avoid taking them, but they were still struggling to find a solution before exceeding the state boundary. After that, the received mean rewards became vibrating around −2, which means many episodes within a batch terminated because of taking terminational actions rather than only exceeding the state boundary. After a longer term of learning, all the RL agents gradually found a way to produce solutions, thus getting the rewards of +6 eventually.

TABLE II
Evaluation Details of the Exemplary Six Processes

	# Evaluations	# Fast Evaluation	# Sizing	Runtime
process1	28,469	174	86	9h:23min
process2	14,952	133	65	8h:22min
process3	10,553	86	38	4h:27min
process4	9,645	89	20	2h:38min
process5	26,053	73	35	3h:48min
Process6	12,685	73	36	3h:15min

C. Analysis of Synthesis Efficiency

Since the time-consuming sizing has to be performed on each generated circuit structure to check its feasibility, the most challenging part of circuit topology synthesis is the evaluation of the produced circuit structures no matter what synthesis strategies are applied. This is especially true for our proposed DRL-based method because training the PGNN needs a huge number of training data. Table II depicts the evaluation details of the above-mentioned six processes. As one can see, almost at least ten thousand evaluations are performed for each process. If the simulation-in-loop sizing is directly applied as the means for evaluation, the synthesis time would be by far unaffordable.

As demonstrated by the number of circuit topologies going to the fast evaluation stage in Table II, after employing the hash table, the number of evaluations has dramatically decreased to just hundreds from at least near ten thousand, which means most of the evaluations were actually performed on the same structures. The reason for this is that a substantial number of episodes produced in the process were repeated, due to batch-size training and the stochastic policy strategy that always makes a decision based on the probability distribution over actions. Therefore, with the help of the hash table, evaluating the generated circuit topologies becomes realizable. However, we are still not clear about whether the circuits that fail the fast evaluation filter are all unable to meet the performance specification through sizing. To test it, we ran the algorithm ten times with distinct filtering requirements. All the fast evaluation failed circuits were recorded for detailed sizing to check the effectiveness of our fast evaluation filter. The average results are depicted in Table III. As one can see, when we set the filtering requirement as dc gain larger than 30 dB, 99% of the fast evaluation failed circuits would fail the detailed sizing

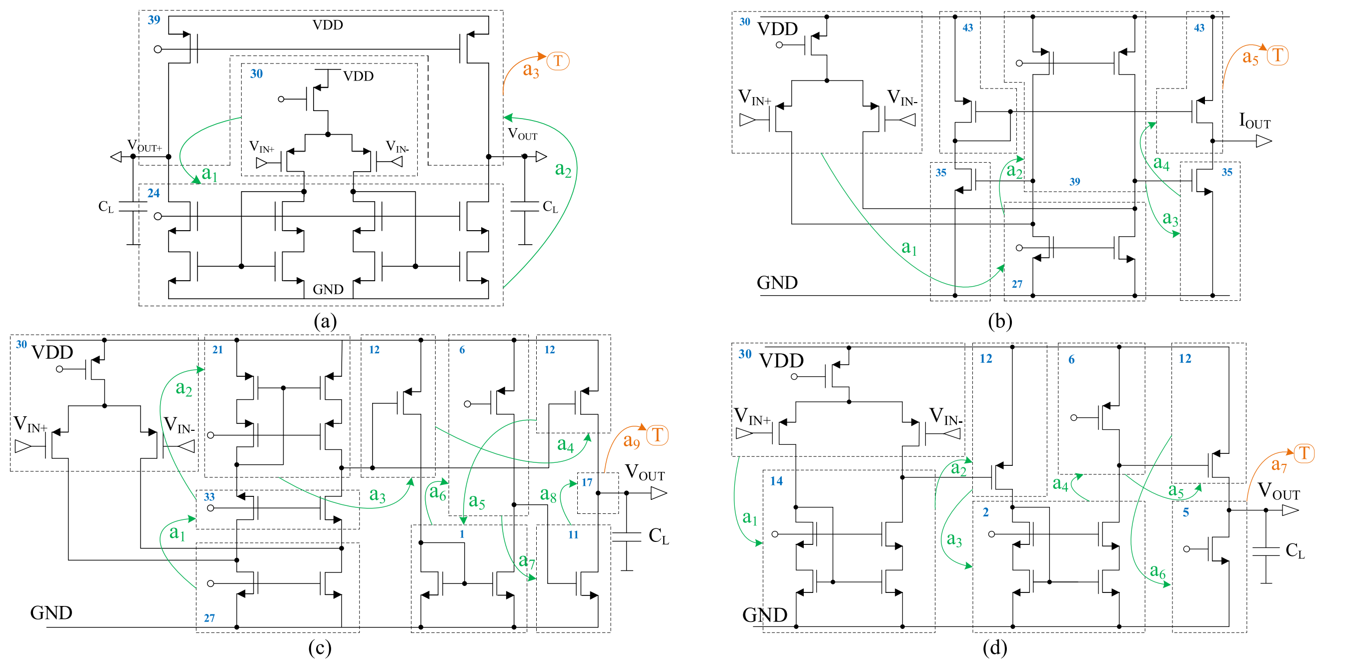
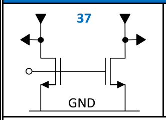
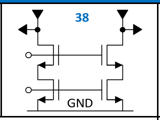
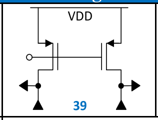
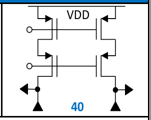
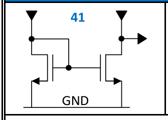
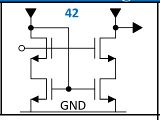
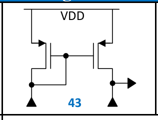
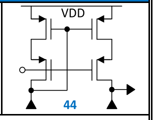


Fig. 11. Example of synthesized circuits. (a) Differential OpAmp. (b) OTA. (c) High gain OpAmp (Av > 100 dB). (d) High unity-gain bandwidth OpAmp (UGB > 1 GHz).

TABLE III
TESTING RESULTS OF THE EFFECTIVENESS OF FAST EVALUATION FILTER

Filtering Requirement	# Fast Evaluation Failure	# Sizing Failure	Correct Rate
> 50 dB	146	101	69%
> 40 dB	138	117	85%
> 30 dB	128	127	99%

TABLE IV
PBBL EXTENSION

Extension of Converter Building Blocks			
			
FI|FI - V|V		FO|FO - V|V	
Current Merger Building Blocks			
			
FI|FI - FO		FO|FO - FO	

check as well. Therefore, in our work, this value is set as the filtering threshold.

As reflected by the number of circuit topologies entering the sizing stage, the fast evaluation filter eliminated at least half of the topologies to be sized, which significantly improved the evaluation efficiency. The runtime of each learning process is provided in the last column of Table II, which varies from 2 to 9 h. It is worth noting that greater number of generated topologies to be sized does not necessarily mean more runtime. For instance, process6 took less runtime than process5 but with more topologies to be sized. This is because some sizing jobs of process5 need more time (i.e., generations) to complete.

D. Analysis of Circuit Topology Synthesis

In this part, besides showing some good circuit topologies that can be synthesized by the proposed work, we will also demonstrate strong reliability and wide applicability of our methodology by using distinct output specifications and performance specifications to synthesize circuits. Figs. 5, 7, and 9 have already depicted the circuit topologies produced by applying the input-output specification as two voltage inputs and one voltage output. To synthesize circuit structures with distinct output specifications, the PBBL defined in Table I has to be extended. Table IV lists some exemplary extension BBs. Among them, the extended converter-type BBs or current merger BBs are used to address output specification as two voltages or one current, respectively. Fig. 11(a) depicts a synthesized differential OpAmp, which is similar to the circuit shown in Fig. 7 except for the last BB selected by a_2. In the differential OpAmp, the 39th BB from the extension BB library is chosen by action a_2. Fig. 11(b) shows an operational transconductance amplifier (OTA) synthesized by our proposed method. It is synthesized by starting with the 30th BB (a pMOS-type differential pair) and taking four actions if excluding the terminational action a_5 shown in a different color. Action a_2 and a_4 select the 39th and 43rd BB from the extension BB library, respectively.

High gain OpAmps and high bandwidth OpAmps always attract more academic and industrial interests. These two categories of circuits with distinct performance specification settings can be readily synthesized by our proposed work.

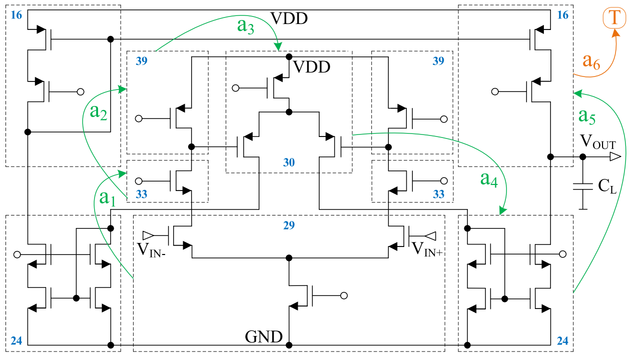


Fig. 12. Novel circuit synthesized.

TABLE V
COMPARISON WITH OTHER CIRCUIT TOPOLOGY SYNTHESIS METHODS

	MOJITO	KMR	FEATS	GCTG	DRL
Result Trustworthy?	yes	yes	yes	yes	yes
Design Search Space Flexible and Controllable?	no	no	yes	yes	yes
Large-Size Circuits Generalizable?	no	no	yes	yes	yes
Creative Circuits Capable?	yes	–	yes	yes	yes
Design Knowledge Automatically Learned?	no	no	no	no	yes
Computation Effort Affordable?	no	no	no	no	yes

In our experiments, the high gain OpAmps are synthesized with the specification of 100-dB *Av*, 60° PM, 10-dB GM, and 10-MHz UGB, while the high bandwidth OpAmps are generated with the specification of 60-dB *Av*, 60° PM, 10-dB GM, and 1-GHz UGB. Fig. 11(c) depicts a synthesized high gain circuit, which is actually a three-stage OpAmp. The synthesis starts with a subcircuit composed of a pMOS-type differential pair and an nMOS-type current mirror, and takes nine actions to construct the circuits. As mentioned in Section V-A, if both normal-match-case2 and normal-match-case3 take place during the construction process, the difference of their occurring time steps will not affect the produced structure if the relative orders of the BBs belonging to the same type of match are preserved. Thereby, in Fig. 11(c), as long as action a_4 is taken after action a_3 but before action a_8, its occurring time step will not affect the synthesized circuit structure. Fig. 11(d) illustrates a synthesized high bandwidth circuit, which contains seven BBs and needs seven actions to build. It can easily achieve more than 1-GHz UGB with power consumption around 0.6 mW, which demonstrates high efficacy of our proposed methodology.

As already demonstrated, even though the state size is only set to 10, our proposed work is still able to synthesize large analog circuits such as three-stage OpAmps. It can be inferred that with a larger size of state and PBBL, more complex circuits can be synthesized by the proposed method due to strong scalability of the NNs. In addition, our proposed work can synthesize innovative circuits. It has three means to potentially generate novel circuits: 1) by enriching PBBL with creative BB; 2) by making creative connections among known BBs; and 3) by combining the two means above. Fig. 12 depicts a novel circuit synthesized by our work via the second means. It is composed of six known BBs, while the creative connections of two differential pairs make this circuit innovative. It can not only reach more than *Av* of 100 dB but also satisfy the other basic performance requirements specified in Section VI-B.

E. Comparison With the State-of-the-Art Methods

So far, we have demonstrated good efficiency, strong reliability, and wide applicability of our proposed DRL-based circuit topology synthesis method. In this part, we will illustrate some comparisons with other state-of-the-art circuit topology synthesis methods. Among these methods, the work of [24] utilizes design knowledge mining and reasoning (KMR) to synthesize circuit topologies while MOJITO [16], FEATS [19], and GCTG [21] use genetic programming, graph composition, and graph decomposition methods to generate circuit topologies, respectively. Table V summarizes the characteristics of the above-mentioned four methods as well as our proposed DRL-based method.

Since all these five methods utilize simulation-in-loop sizing to verify the produced circuit topologies, their synthesized solutions are trustworthy. There is no means for MOJITO to control its design search space, and its defined crossover and mutation operations are only workable for a narrow range of circuits, typically one-stage or two-stage OpAmps shown in its experimental results. Since KMR has to mine the design knowledge beforehand and then apply it to synthesize circuits in a reasoning way, it is difficult to flexibly extend its design search space and freely generalize its synthesis to another class of circuits that have not been learned. FEATS and GCTC are able to flexibly extend the design search space to address large-size circuit designs by extending the defined BB library or increasing the maximum number of BBs allowed for synthesis. Similar to them, our proposed DRL-based method can flexibly extend the design search space and easily generalize to large-size circuit design by extending PBBL or increasing state size.

Except for KMR, all the other methods claim that they are able to synthesize less-known circuits. The design knowledge is encoded in the predefined crossover/mutation operations, composition rules, and decomposition rules in the works of MOJITO, FEATS, and GCTC, respectively. Thus, there is no knowledge learning involved in these works. Since the design knowledge learning process of KMR involves a great amount of human intervention, it is not deemed as automatic learning. However, our proposed work utilizes DRL to automatically learn design knowledge, which is stored in the NN.

Due to the stochastic-driven synthesis, MOJITO needs to evaluate around 101 904 topologies in the process to finally produce 512 unique topologies, which took seven days. Thereby, MOJITO suffers from almost unaffordable computation effort to synthesize circuits. KMR has to go through a

TABLE VI
COMPARISON AMONG FEATS, GCTG, AND DRL BY USING PBBL

Topology Size	Number of the Unique Topologies Generated		
	FEATS	GCTG	DRL
5	1,056	832	20
6	6,780	6,152	43
7	43,175	45,776	61
8	–	–	96
9	–	–	127

quite timing-consuming data mining process because the similarity of circuits is extracted by comparing them in pairs. Due to the brute-force explorative nature of both FEATS and GCTC, their computation effort is also unaffordable especially if the size of the BB library is large. Table VI illustrates the comparison among FEATS, GCTC, and our proposed DRL by using the same BB library as listed in Table I (i.e., the PBBL) and the same input–output specification (i.e., two voltage inputs and one voltage output). As one of the table headers in Table VI, *Topology Size* refers to the maximum block number, maximum leave number, and maximum state size allowed in FEATS, GCTC, and DRL, respectively. Since the DRL process features stochastic nature, its results in Table VI are the average outputs from ten runs of our proposed algorithm.

As one can see, when the topology size is 5, FEATS and GCTC generate around 1000 unique circuit topologies after applying their isomorphism checks, whereas our proposed DRL only produces 20 unique circuit topologies during the synthesis process. It is worth mentioning that such a significant number contrast should not be deemed as capability inferiority of circuit topology construction for the DRL method. Instead by comparing the nature of the three methods above, we can see that FEATS and GCTC are only dedicated to the construction of circuit topologies in the brute-force way without concerning the quality of the generated topologies. Thus, a large majority of their generated circuit topologies turn to be fruitless. However, our proposed DRL method can smartly learn from the construction experience and only selectively generate increasingly more promising circuit topologies. More important, the number of the unique topologies produced by FEATS and GCTC grows exponentially in terms of the allowed topology size. For instance, as listed in Table VI, when the topology size increases to only 7 (not saying 8 and 9 that are unsolvable per se), FEATS and GCTG would generate more than 40 000 unique circuit structures for evaluation, which definitely leads to a barrier to their industrial applications in practice. However, this topology explosion issue can be readily addressed by our proposed DRL methodology, thanks to strong scalability of the NNs. As shown in Table VI, the number of the unique circuit topologies generated by DRL only increases a bit when the topology size grows.

VII. CONCLUSION

In this article, we presented a novel DRL-based method to synthesize analog circuits in a smart trial-and-error process that mimics the self-learning manner of humans, where the learned intelligence is stored in an NN. It is able to manage large-size and innovative circuit synthesis. It also has wide applicability to deal with distinct input–output and performance specifications. Thanks to simulation-in-loop involvement and other speed-up schemes, it has the ability to accurately and efficiently verify the output solutions. Compared with the other state-of-the-art circuit topology synthesis methods, it can not only address their commonly known shortcomings but also achieve the least computation effort by synthesizing a solution with a learning process that consumes only several hours on average, which is a significant improvement to industrial applications. Furthermore, another big advantage of our proposed DRL-based method is that the trained model can always be reused, which is able to provide a solution with the same input within 1 s.

As our future research work, we intend to further develop advanced design rules for tackling the compensation for multistage structures. To support this, we also need to develop a fundamental method to evaluate the contribution from compensation to the fitness in the RL process. Moreover, we like to extend our scope to larger and more complex analog circuits.

REFERENCES

[1] Z. Zhao and L. Zhang, "Efficient performance modeling for automated CMOS analog circuit synthesis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 11, pp. 1824–1837, Nov. 2021.
[2] L. Zhang and Z. Liu, "Directly performance-constrained template-based layout retargeting and optimization for analog integrated circuits," *Integr. VLSI J.*, vol. 44, no. 1, pp. 1–11, 2011.
[3] G. Shomalnasab, H. M. Heys, and L. Zhang, "Analytic modeling of interconnect capacitance in submicron and nanometer technologies," in *Proc. IEEE Int. Symp. Circuits Syst.*, Beijing, China, 2013, pp. 2553–2556.
[4] T. Liao and L. Zhang, "An LDE-aware g_m/I_D-based hybrid sizing method for analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1511–1524, Aug. 2021.
[5] R. A. de Lima Moreto, C. E. Thomaz, and S. P. Gimenez, "Gaussian fitness functions for optimizing analog CMOS integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 10, pp. 1620–1632, Oct. 2017.
[6] O. Garitselov, S. P. Mohanty, and E. Kougianos, "Fast-accurate non-polynomial metamodeling for Nano-CMOS PLL design optimization," in *Proc. Int. Conf. VLSI Des.*, Hyderabad, India, 2012, pp. 316–321.
[7] O. Okobiah, S. Mohanty, and E. Kougianos, "Fast design optimization through simple Kriging metamodeling: A sense amplifier case study," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 4, pp. 932–937, Apr. 2014.
[8] Y. Yang *et al.*, "Smart-MSP: A self-adaptive multiple starting point optimization approach for analog circuit synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 3, pp. 531–544, Mar. 2018.
[9] W. Lyu *et al.*, "An efficient Bayesian optimization approach for automated optimization of analog circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1954–1967, Jun. 2018.
[10] G. Stehr, H. E. Graeb, and K. J. Antreich, "Analog performance space exploration by normal-boundary intersection and by Fourier–Motzkin elimination," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 10, pp. 1733–1748, Oct. 2007.
[11] Z. Zhao, T. Liao, and L. Zhang, "Fast performance evaluation for analog circuit synthesis frameworks," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, 2018, pp. 1–5.
[12] R. Harjani, R. A. Rutenbar, and L. R. Carley, "OASYS: A framework for analog circuit synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 12, pp. 1247–1266, Dec. 1989.
[13] G. V. D. Plas *et al.*, "AMGIE-A synthesis environment for CMOS analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1037–1058, Sep. 2001.

[14] S. E. Sorkhabi and L. Zhang, "Automated topology synthesis of analog and RF integrated circuits," *Integr. VLSI J.*, vol. 56, pp. 128–138, Jan. 2017.

[15] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 596–607, Oct. 2007.

[16] T. McConaghy, P. Palmers, M. Steyaert, and G. G. E. Gielen, "Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks," *IEEE Trans. Evol. Comput.*, vol. 15, no. 4, pp. 557–570, Aug. 2011.

[17] J. Slezak and J. Petrzela, "Evolutionary synthesis of cube root computational circuit using graph hybrid estimation of distribution algorithm," *Radioengineering*, vol. 23, no. 1, p. 549, 2014.

[18] Ž. Rojec, Á. Bűrmen, and I. Fajfar, "Analog circuit topology synthesis by means of evolutionary computation," *Eng. Appl. Artif. Intell.*, vol. 80, pp. 48–65, Apr. 2019.

[19] M. Meissner and L. Hedrich, "FEATS: Framework for explorative analog topology synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 2, pp. 213–226, Feb. 2015.

[20] Z. Zhao and L. Zhang, "Graph-grammar-based analog circuit topology synthesis," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Sapporo, Japan, 2019, pp. 1–5.

[21] Z. Zhao and L. Zhang, "An automated topology synthesis framework for analog integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4325–4337, Dec. 2020.

[22] C. Ferent and A. Doboli, "Novel circuit topology synthesis method using circuit feature mining and symbolic comparison," in *Proc. Des. Autom. Test Eur. Conf. Exhibit.*, Dresden, Germany, 2014, pp. 1–4.

[23] C. Ferent and A. Doboli, "Symbolic matching and constraint generation for systematic comparison of analog circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 4, pp. 616–629, Apr. 2013.

[24] F. Jiao, S. Montano, C. Ferent, A. Doboli, and S. Doboli, "Analog circuit design knowledge mining: Discovering topological similarities and uncovering design reasoning strategies," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 7, pp. 1045–1058, Jul. 2015.

[25] H. Li, X. Liu, F. Jiao, A. Doboli, and S. Doboli, "InnovA: A cognitive architecture for computational innovation through robust divergence and its application for analog circuit design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 10, pp. 1943–1956, Oct. 2018.

[26] Z. Zhao and L. Zhang, "Deep reinforcement learning for analog circuit sizing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seville, Spain, 2020, pp. 1–5.

[27] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. Des. Autom. Test Eur. Conf. Exhibit. (DATE)*, Grenoble, France, 2020, pp. 490–495.

[28] H. Wang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. ACM/IEEE Des. Autom. Conf. (DAC)*, San Francisco, CA, USA, 2020, pp. 1–6.

[29] M. Ahmadi and L. Zhang, "Analog layout placement for FinFET technology using reinforcement learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Daegu, South Korea, 2021, pp. 1–5.

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[31] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[32] L. Zhang, R. Raut, L. Wang, and Y. Jiang, "Analog module placement realizing symmetry constraints based on a radiation decoder," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Hiroshima, Japan, 2004, p. I481.

[33] T. Liao and L. Zhang, "Layout-dependent effects aware g_m/I_D-based many-objective sizing optimization for analog integrated circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, 2018, pp. 1–5.

[34] A. A. I. Ahmed and L. Zhang, "Fast parasitic-aware synthesis methodology for high-performance analog circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seoul, South Korea, 2012, pp. 2155–2158.

Zhenxin Zhao (Member, IEEE) received the M.Sc. degree in computer engineering from the Memorial University of Newfoundland, St. John's, NL, Canada, in 2016, where he is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering.

His current research interests include analog circuit topology synthesis, analog placement, analog circuit sizing, and machine learning-assisted circuit analysis and design automation.

Lihong Zhang (Member, IEEE) received the Ph.D. degree in electrical engineering from the Otto-von-Guericke University of Magdeburg, Magdeburg, Germany, in 2003.

He was a Postdoctoral Research Associate with Concordia University, Montreal, QC, Canada; Dalhousie University, Halifax, NS, Canada; and the University of Washington, Seattle, WA, USA. He is currently a Full Professor with the Department of Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, NL, Canada. His current research interests include very large-scale integration computer-aided design, mixed-signal integrated system/circuit design, microelectromechanical systems design and design automation, wireless sensor networks, microfluidics and biosensors, and microprocessor-based instrumentation for ocean and biomedical applications.