

Simulator-Driven Deep Reinforcement Learning for Analog Circuit Design

Felicia B. Guo, Ken T. Ho, Andrei Vladimirescu, Borivoje Nikolić

Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA

{felicia_guo, ken_ho, andreiv, bora}@berkeley.edu

Abstract—This work addresses the use of reinforcement learning in the design of analog and mixed-signal (AMS) circuits. With recent advanced angstrom-technology-nodes adding new complexities, this highly manual process has grown increasingly challenging and less aligned with conventional design intuition. The presented approach modifies circuit topologies at the transistor-level to meet design requirements. We present, for the first time, a deep reinforcement learning (RL) framework capable of generating novel circuit topologies by using graph encodings for targeted specifications, starting from a minimal expert design and a user-specified testbench. To highlight the capabilities of the approach, we demonstrate the topological modification and expansion of incomplete sub-circuits to satisfy user-provided performance for three different types of circuits: 1) a ring oscillator, 2) a comparator, and 3) an operational transconductance amplifier. Our results demonstrate that our method is capable of generating previously unseen topologies that reach user-defined performance targets. In each design case, 100% of generated circuit netlists are correct by construction and over 90% of generated circuits demonstrate intended functionality and targeted performance when simulated with commercial tools.

I. INTRODUCTION

Analog/mixed-signal (AMS) circuits play a critical role in modern integrated circuit systems. As system architectures scale-up and scale-out through chiplets, advanced packaging, and interconnect, rapid AMS implementation is necessary to meet demanding design specifications. Despite the growth in AMS system complexity, many underlying analog design processes remain unchanged. Manual methodologies continue to dominate critical tasks such as circuit topology selection and design, component sizing, and physical layout.

Multiple frameworks have approached AMS design automation through generator-based methodologies. In these paradigms, design processes for different circuit classes are encoded into generation scripts using standardized APIs [1], [2]. Broadly, this approach aims to formalize design intuition into reproducible code. However, because these frameworks depend on extensive rules and intricate routines, they inherit some of the same limitations encountered by human designers. In advanced nodes, nonlinearities and discontinuities often challenge brittle intent-encoding flows. Given these difficulties, modern machine learning (ML) offers a compelling alternative. Recent advances in adjacent domains have demonstrated the effectiveness of deep learning solutions when compared to solely rule-based methodologies [3], [4].

Previous ML-based approaches have largely focused on circuit sizing. Given a fixed circuit topology, these methods adjust component parameters (dimensions and multiplicity, for

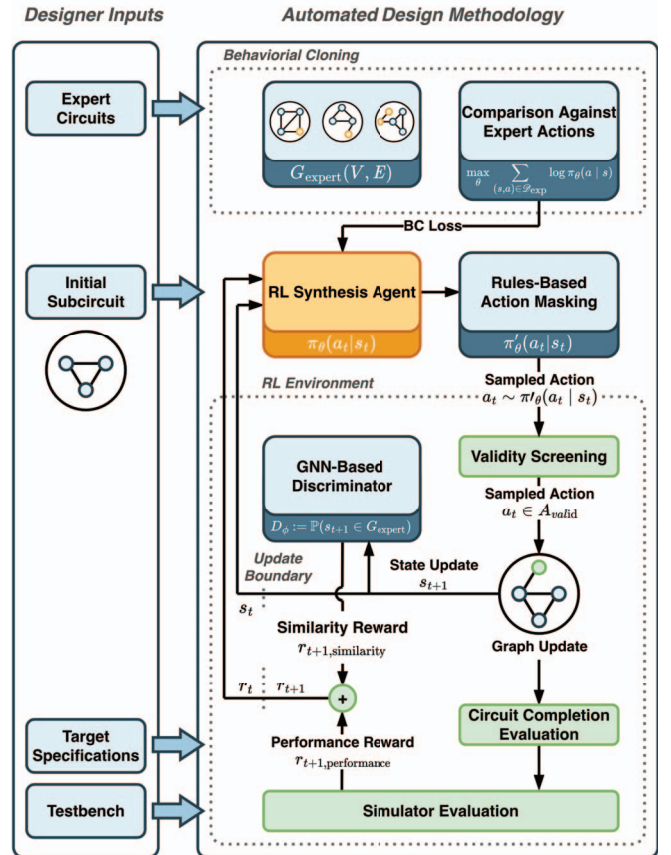


Fig. 1: Block diagram of the transistor-level design framework. Inputs specified by the designer are shown on the left; automated algorithmic stages are shown on the right.

example) to meet updated performance specifications. AutoCkt frames this within an RL context [5], while other frameworks employ Bayesian optimization [6]–[8]. More recent efforts have explored the synthesis of new topologies. LaMAGIC [9] and AnalogCoder [10] use LLMs to generate circuits as text, while CktGNN [11] applies graph neural nets (GNNs) to synthesize block-level operational amplifiers. AnalogGenie further demonstrates transistor-level design through a transformer-based model [12]. Critically, these methods do not directly train on performance-based feedback from circuit simulators, limiting their capability to learn consistent methods of achieving new design specifications.

In this paper, we present a specification-driven analog syn-

thesis methodology for transistor-level topology generation (Fig. 1). Our approach combines reward-driven stochastic exploration with ground-truth circuit simulator feedback to meet target specifications. Within this framework, we introduce three techniques that further enable automated analog design: 1) rule-based masking to prevent invalid actions, 2) generalized reward design for functional design generation, and 3) behavioral cloning for implicit biasing. Through rule-based masking, the framework further ensures that all generated circuit netlists are syntactically correct by construction.

II. DESIGN FRAMEWORK

A. Preliminaries

To generate a new circuit, this framework takes in 1) design specifications, 2) netlists of expert designs, 3) a single netlist of a seed subcircuit (e.g., differential pair) and 4) a commercial simulator testbench. An **expert topology** is a conventional design used to guide initial synthesis. Experts are composed of generic components, without technology-specific or sizing details. From this experiment construction, our framework automates the evaluation and modification of expert topologies.

Operationally, the circuits within the framework are represented as compact graphs $G = (V, E)$, similar to [13]. In this representation, each component and net in a circuit is treated as a node in the graph. Edges connect nodes, with feature vectors indicating the types of terminals they join. More formally, for nodes $u, v \in V$:

$$F_u = [\mathbb{1}_{\text{net}}, \mathbb{1}_{\text{PFET}}, \mathbb{1}_{\text{NFET}}, \mathbb{1}_{\text{resistor}}, \mathbb{1}_{\text{capacitor}}, \mathbb{1}_{\text{inductor}}, \mathbb{1}_{\text{input}}, \mathbb{1}_{\text{output}}, \mathbb{1}_{\text{VDD}}, \mathbb{1}_{\text{GND}}] \quad (1)$$

$$E_{(u,v)} = [\mathbb{1}_{\text{bulk}}, \mathbb{1}_{\text{drain}}, \mathbb{1}_{\text{gate}}, \mathbb{1}_{\text{source}}, \mathbb{1}_{\text{passive } +}, \mathbb{1}_{\text{passive } -}, \mathbb{1}_{\text{passive bulk}}] \quad (2)$$

Here, F_u encodes standard component types (e.g., PFET, NFET, etc.), general nets, and four special nets (VDD, VSS, input, output nodes). The graph representation is illustrated in Fig. 2.

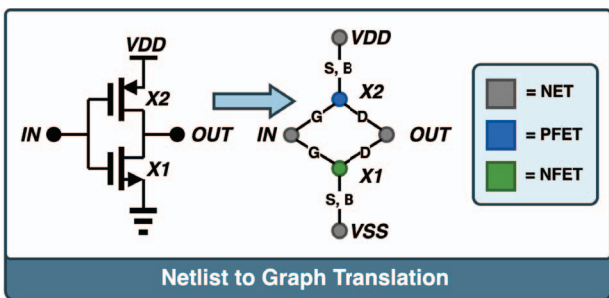


Fig. 2: An inverter is converted into a compact graph representation. Edge features are simplified for illustrative clarity.

B. Reinforcement Learning Formulation

Reinforcement learning (RL) is a machine learning paradigm in which an agent interacts with an environment (E) modeled as a Markov decision process ($\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{S}'$) [14]. At each timestep

t , an agent observes a state $s_t \in \mathcal{S}$, and takes an action $a_t \in \mathcal{A}$ based on a policy $\pi_\theta(a_t|s_t)$. More explicitly, policy π_θ defines the probability of selecting an action a_t given s_t . Based on a_t , the agent receives a reward $r_t = R(a_t, s_t)$. Through repeated rollouts (sequences of interactions), the agent learns an optimal policy $\pi_\theta(a_t|s_t)$ maximizing long-range reward accumulation.

Our work introduces a novel formulation that enables the synthesis of new topologies through the sequential addition of circuit components. This approach is inspired by generative RL applications in other fields [15]. The following described process is fully illustrated in Fig. 1.

State and Action Spaces: In this framework, state s_t is represented by a partially constructed circuit graph, referred to as a **circuit construct**. At $t = 0$, observed state s_0 is the expert-provided seed subcircuit exactly (e.g., differential pair). At each action step, the RL agent takes action a_t and decides whether to continue or conclude the synthesis. Action a_t is defined as the addition of an edge connection between an existing node and either 1) another existing node or 2) a newly added node. Nodes that can be legally added are referred to as **scaffold nodes**.

Validity Constraints: The validity of a_t is determined by environment E . E enforces rules that must be met before an action can be taken to change the circuit construct.

- 1) The first node must already exist in the graph construct. The second must either exist or be a valid scaffold node.
- 2) Component nodes can only connect to net nodes and vice versa.
- 3) Every terminal of a component node may only be connected to a single net.
- 4) If concluding synthesis, all component terminals must be connected.

These rules mask the full action space \mathcal{A} , reducing the number of possible actions a_t to reflect realistic design actions.

Reward Design: The reward design consists of 1) a validity reward, 2) a similarity reward, and 3) a circuit performance reward. The validity reward encourages actions adhering to validity constraints. The similarity reward discourages actions that lead to significant deviations from expert designs. The framework assesses the reward through a GINE-based (graph isomorphism net with edge features [16]) discriminator network that outputs the probability that a circuit construct input is part of an expert design. Finally, the circuit performance reward assesses simulator-verified performance of a fully-generated circuit. Algorithm 1 details reward assessment.

C. Policy Training

In this framework, the RL agent's policy π_θ is realized as another GINE-based network and is trained by using proximal policy optimization (PPO) [17]. PPO maximizes expected rewards while using clipped objectives to prevent destabilizing policy updates. Critically, behavioral cloning (BC) is employed to initially guide the agent towards expert trajectories by

Algorithm 1: Reward Calculation

Given: Circuit construct graph s_t , action a_t , sequence step t , discriminator network D_ϕ , performance specifications P with weights $\{w_s\}_{s \in P}$

Result: Reward r

if a_t is not valid **then**

$s_{t+1} \leftarrow s_t$
 return $R \leftarrow -2$

else

 Add action a_t to s_t to obtain s_{t+1}

$r_{\text{similarity}} \leftarrow \begin{cases} 1, & \text{if } D_\phi(s_{t+1}) > 0.5 \\ -1, & \text{otherwise} \end{cases}$

if a_t indicates done **or** $t >$ timestep threshold **then**

if design does not have floating nodes **then**

if functional design **then**

$r_{\text{functional}} \leftarrow 30$

foreach $s \in P$ **do**

$v \leftarrow$ simulated performance

$r_{\text{opt}} \leftarrow w_s * \begin{cases} 1 - \frac{v-s}{e} & \text{if target, } e \text{ is error} \\ \frac{v-s}{v-3s} & \text{if minimize} \\ \frac{v-s}{v+s} & \text{if maximize} \end{cases}$

$r_{\text{performance}} \leftarrow r_{\text{functional}} + r_{\text{opt}}$

if $r_{\text{opt}} > 0$ **then**

$r_{\text{performance}} \leftarrow r_{\text{perf}} + 10$

else

$r_{\text{performance}} \leftarrow 3$

else

$r_{\text{performance}} \leftarrow -2$

$r \leftarrow r_{\text{similarity}} + r_{\text{perf}}$

return r

shaping action distributions to match that of an expert. An expert trajectory is defined as a sequence of action-state pairs $\tau_e = \{(a_0, s_0), (a_1, s_1) \dots (a_t, s_t)\}$ that maps an initial seed subcircuit to the previously discussed conventional, expert topology (human-designed). As with the expert topologies, expert trajectories operate over generic components without technology and sizing details.

By employing behavioral cloning, the RL agent is able to reproduce circuits resembling conventional topologies. As synthesis progresses, the framework tapers the influence of behavioral cloning to encourage agent exploration (e.g., adding new components). Algorithm 2 details policy training.

D. Framework Single Action Sequence

This section illustrates the RL formulation and policy with a single action sequence, shown in Fig. 3. The seed subcircuit is first converted into a graph (Panel 0). Guided by its policy, the agent selects node 8 (NFET) as the target for expansion (Panel 1). Nodes 5, 6, and 7 are masked since all their terminals are already connected. By the framework’s validity rules, when a component node is selected, it must be connected to either 1) an existing net node or 2) a newly created net node. The agent then selects output net node 3 and assigns the edge feature as drain, the only unconnected terminal of the component (Panels 2–3). Finally, the agent evaluates termination and chooses to continue the synthesis process (Panel 4).

Algorithm 2: Policy Training Iteration

Given: Policy π_θ with parameters θ , training iteration k

1. Collect policy trajectories:

 Generate a batch of circuit construction trajectories by rolling out π_θ in the environment

2. Compute PPO objective:

$$\mathcal{L}^{\text{PPO}}(\pi_\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\pi_\theta, \pi_{\theta_k}) A_t^{\pi_{\theta_k}}, \text{clip}(\rho_t(\pi_\theta, \pi_{\theta_k}), 1 - \epsilon, 1 + \epsilon) A_t^{\pi_{\theta_k}} \right) + c \mathcal{H}[\pi_\theta](s_t) \right]$$

where $\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$ is the importance ratio, $A_t^{\pi_{\theta_k}}$ is the advantage estimate calculated from rewards in trajectories, and \mathcal{H} is the entropy bonus

3. Sample expert demonstrations:

 Sample a batch of expert state-action pairs $\mathcal{D}_{\text{exp}} = \{(s, a)\}$ using breadth-first search (BFS) originating from GND node

4. Add behavioral cloning (BC) objective:

$$\mathcal{L}^{\text{BC}}(\pi_\theta) = \sum_{(s,a) \in \mathcal{D}_{\text{exp}}} \log \pi_\theta(a|s)$$

5. Joint optimization:

$$\theta^* = \arg \max_{\theta} \mathcal{L}^{\text{PPO}}(\pi_\theta) + \lambda_0 \lambda_1^k \mathcal{L}^{\text{BC}}(\pi_\theta)$$

where λ_0, λ_1 control weighting of BC loss

III. EXPERIMENTAL RESULTS

We demonstrate our methodology with design tasks involving three circuit classes. Prior to deployment, the agent undergoes one-time pretraining with behavioral cloning on a full expert dataset, consisting of approximately 1100 circuit topologies (including a subset of designs from [12]). This process enables the agent to learn a general structural representation of circuits. During policy training for specific design tasks, behavioral cloning is further applied with a smaller set of expert topologies to provide task-specific guidance (Algorithm 2). In addition, for each design task, the discriminator is trained once to classify a small set of circuit constructs sampled from expert designs (positive samples) and topologies generated by the pretrained agent (negative samples). The policy and discriminator networks each use separate backbones of three GINE layers with 64-dimensional hidden embeddings, followed by fully-connected layers for action selection and discriminator classification, for a total of approximately 212k parameters.

To quantitatively evaluate performance across design tasks, the four following metrics are defined. Simulation validity and specification fulfillment are computed based on satisfaction of raw performance metrics rather than proxy figures of merit (FoM). All simulations use the Skywater 130nm process.

- 1) Netlisting Validity:** Percentage of generated designs that pass circuit netlisting. This metric captures syntactic correctness for generated designs.
- 2) Simulation Validity:** Percentage of designs that simulate successfully with sufficient functionality to enable meaningful evaluation of analysis metrics (e.g., frequency, GBW). Designs need not fulfill design specifications.

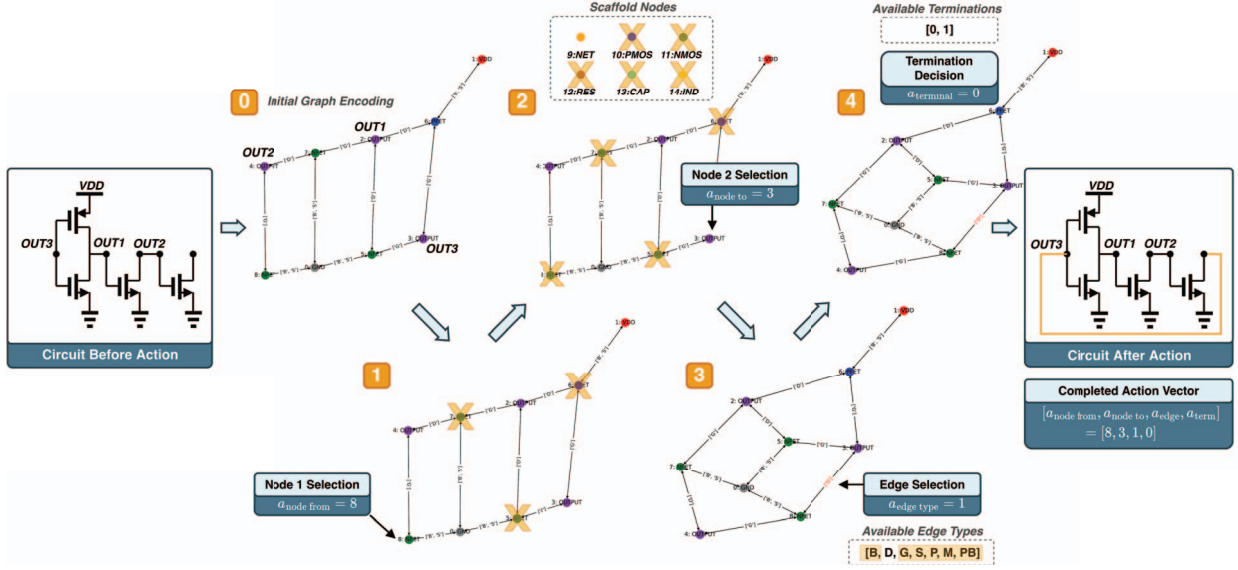


Fig. 3: A single action sequence, as detailed in Section 2D. (0) Circuit parsed to graph; (1) first node selected; (2) second node selected from existing and scaffold spaces; (3) terminal selected for connection; (4) termination decision. Invalid options crossed.

- 3) **Specification Fulfillment:** The percentage of generated designs within 10% of targets or meeting/exceeding maximum or minimum design specifications.
- 4) **Novelty:** Percentage of new, generated designs absent from the expert dataset. Novel designs do not need to satisfy validity or fulfillment metrics (per prior work).

A. Ring Oscillator

In this task, the framework is applied to a design of a ring oscillator (RO), targeting a 4.75 GHz operational frequency with an even duty cycle and minimal average power consumption. A power constraint of 500 μW is imposed to prevent unbounded designs. The process is seeded with four transistors sampled from a standard RO (shown in black, Fig. 4a). For reward calculation, a weight of 15 was applied to frequency, and a weight of 1 was applied to the other specifications. Additionally, the agent is provided with two generic expert structures: a 3-stage and a 7-stage inverter-based RO.

To evaluate performance, the agent is provided a testbench measuring frequency, duty cycle and average power draw. Voltage swing is computed at three separate nodes to differentiate valid designs from spurious ones.

Fig. 4a shows a solution in which the agent reconstructed two expert transistors (orange) and discovered five new transistors (blue) to meet specifications. Transistors XP0 and XP3 increase the multiplicity of the PFETs in each delay stage, improving the oscillator’s duty cycle by compensating for the lower PFET mobility constant. Transistors XP0 and XP2 operate in cutoff or deep-triode regions, loading each stage with C_{gs} capacitances. Transistor XP4 operates entirely in cutoff (gate connected to VDD), providing parasitic C_{gd} , C_{gs} capacitances to load stages.

The simulated transient waveform is shown in Fig. 4b. While there is some imbalance in the duty cycle of OUT1, this is subsequently corrected in the OUT2 and OUT3 stages to create an even 49.84% duty cycle measurement on OUT3.

Across training iterations, the agent progressively learned to generate novel designs. The novelty curve indicates that the agent initially recreated expert designs before exploring topologies with new loading schemes. With a fully-trained agent, 90.1% of generated circuits were valid in simulation. Because oscillation frequency is highly sensitive to capacitive loading, successful designs required precise load design, and only 13.6% satisfied required specifications (Fig. 4c).

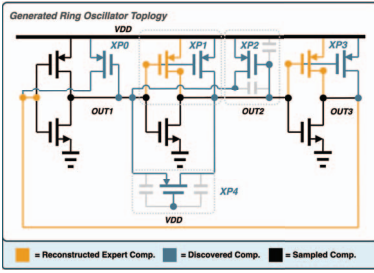
B. Comparator

In this task, the framework is applied to design a comparator capable of driving a 100fF capacitive load with a maximum input-referred noise of 0.7 mVrms, maximum delay of 600 ps, and power constraint of 200 μW . For reward calculation, each specification was assigned a weight of 15. The process is seeded with seven transistors constituting a differential pair, tail, and basic latch (shown in black, Fig. 4d). Additionally, the agent is provided with a StrongARM latch as an expert structure.

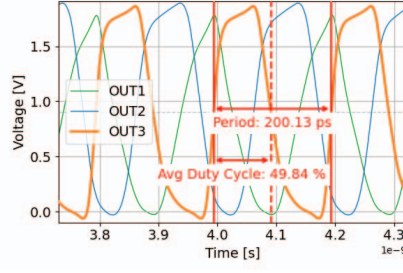
To evaluate performance, the agent is provided a testbench that measures delay, input-referred noise (1mV differential input), and average power draw. Input offset is computed to differentiate valid designs from spurious ones, with a maximum of 10mV imposed. During construction, each transistor addition action creates a unit device of fixed dimensions, sized practically given the capacitive load.

Fig. 4d shows a solution in which the agent reconstructs the output precharge switches (orange) before introducing a new set of switches (blue). These switches, labeled XP0 and XP1, are equalization switches that force the outputs to the same potential during the precharge phase. The simulated transient waveform of the comparator is shown in Fig. 4e.

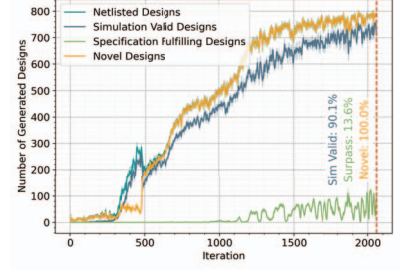
Training dynamics are shown in Fig. 4f. With a fully-trained agent, 99.1% of the generated circuits were valid in simulation and met fulfilled specifications, demonstrating the effectiveness of this framework on non-linear circuit design tasks.



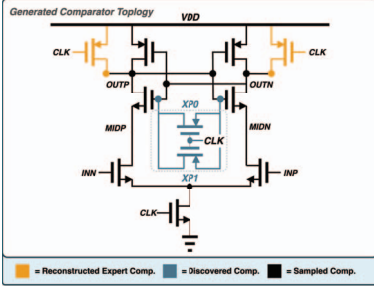
(a) Generated RO design. 4.996 GHz, 49.84% duty cycle, 428 μ W power consumption.



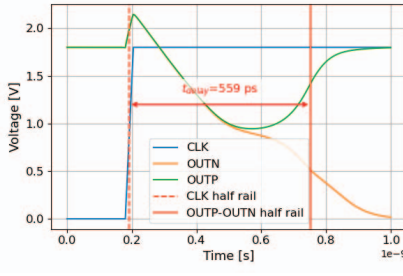
(b) Transient waveform showing outputs from nets OUT1, OUT2, and OUT3 corresponding to Fig. 4a.



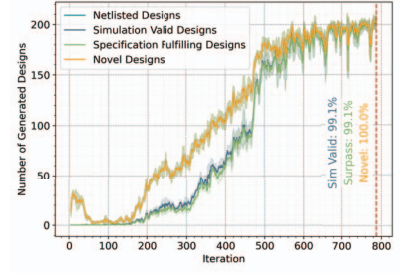
(c) Training dynamics of the RO design task.



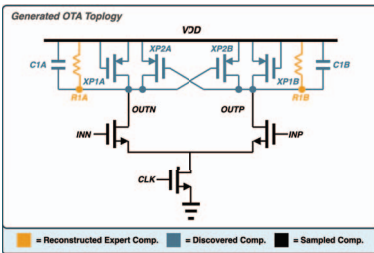
(d) Generated comparator design. 0.59mVrms input referred noise, 559 ps delay, 184 μ W of power.



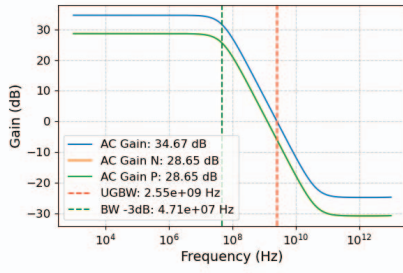
(e) Transient waveforms showing comparator output transition corresponding to Fig. 4d.



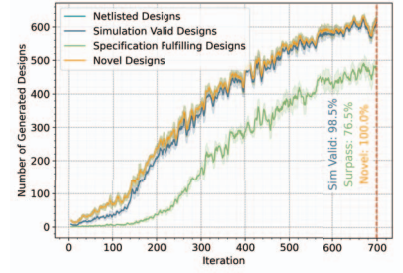
(f) Training dynamics of the comparator design task.



(g) Generated OTA design. 34.67 dB gain, 47.1 MHz 3dB bandwidth, 713 μ W power consumption.



(h) AC differential gain and single sided gain of the OTA corresponding to Fig. 4g. OTA gain on two sides overlap.



(i) Training dynamics of the OTA design task.

Fig. 4: Overview of design task outputs. Figs. (a, d, g) show generated designs. Initial samples are in black. Components belonging to expert design(s) are in orange. Agent discovered components are in blue. Figs. (b, e, h) show the corresponding testbench waveform visualization. Figs. (c, f, i) show training dynamics for each task. Over iterations, the number of designs that are netlisted, simulation valid, satisfying design targets, and novel are plotted. Iterations are shown up to convergence. Different tasks converge at different speeds as is commonly expected from RL deployment.

C. Operational Transconductance Amplifier

In this task, the framework is applied to design a single-stage differential operational transconductance amplifier (OTA) driving a 100 fF capacitor load with a minimum AC gain of 17 dB, minimum 3 dB bandwidth of 10 MHz, maximum gain mismatch of 2 dB, and power constraint of 10 mW. For reward calculation, each specification was assigned a weight of 15. The process is seeded with three transistors, constituting a differential pair and tail source (shown in black, Fig. 4g). Additionally, the agent is provided with three generic expert structures: five-transistor OTA with common-mode feedback resistors, five-transistor OTA with parallel resistive load, and

five-transistor OTA with series resistive load. For this design task, the agent takes duplicate actions on each side of the differential half-circuit to better fulfill half-circuit design intent.

To evaluate performance, the agent is provided a testbench measuring AC gain and mismatch, 3dB bandwidth, and average power draw. Input bias is set to 1 V and other biases are set to half-rail. As before, transistor addition creates a unit device of fixed dimensions sized practically given the capacitive load. While fixed bias voltages are not ideal for headroom, our framework can still perform under this constraint, highlighting the broad applicability of the RL framework. This constraint is also imposed by other state-of-the-art (SoTA) frameworks [10].

Fig. 4g shows a solution in which the agent reconstructs two

expert resistors (orange) and discovers four new transistors and two new capacitors (blue) to meet design specifications. Cross-coupled PFETs XP2A and XP2B and the sampled differential input NFETs contribute to the OTA transconductance. The OTA output resistance is dominated by resistors R1A and R1B. We note that the other components (C1A, C1B, XP1A, XP1B) do not contribute to the performance of this OTA design. Since only netlist rules are enforced, incidental reward signals can cause extraneous components to appear in generated circuits.

With a fully trained agent, 98.5% of these generated designs were functional and 76.5% passed design targets by the provided testbench, as shown in Fig. 4i.

D. Comparative Evaluation

We benchmark this framework against available, open-source SoTA analog synthesis frameworks using the same RO, comparator, and OTA examples. We now briefly summarize these frameworks and their underlying paradigms. AnalogCoder uses commercial LLMs with domain-specific prompt-engineering to generate circuits. For evaluation, the GPT-4o version is used (GPT-3.5 included for reference). AnalogGenie uses a transformer-based model coupled with reinforcement learning from human feedback (RLHF) to produce expert-aligned netlists. Table I provides a qualitative comparison.

To enable standardized evaluation, all frameworks are provided with comparable testbenches, technology models, and expert circuits. Variations exist due to differences in framework environments and tooling. Generated designs from each were assessed for raw circuit performance in SPICE-based testbenches with ground-truth simulators.

Netlisting Validity: As shown in Table II, AnalogCoder and AnalogGenie achieved strong netlisting validity across tasks, averaging 81.1% and 93.2%, respectively. AnalogCoder utilized careful prompt-engineering and ran multiple attempts per iteration to reduce code-generation errors. AnalogGenie’s transformer-based model learned circuit connectivity patterns from a large topology dataset. Through formal netlisting rule checks, our framework achieves 100% netlisting validity.

Simulation Validity/ Specification Fulfillment: As shown in Table II, AnalogCoder achieved an average simulation validity rate of 40.0% and specification fulfillment rate of 16.7%. This is consistent with prior observations in [10] for challenging design tasks. General-purpose LLMs lack in-depth understanding of circuit behavior, limiting their efficacy in iterative, complex multi-objective domain tasks. For AnalogGenie, the pretraining methodology from [12] was followed to successfully reproduce reported pretrained model performance. Consistent with the original work, RLHF was applied on all tasks using newly constructed datasets. However, this did not yield expected improvements. As amplifier designs were strongly represented in the released pretraining data, the pretrained model performed reasonably on the OTA task despite lacking RLHF. We report these as baseline metrics, noting potential RLHF-tuned performance may be stronger. RO and comparators were underrepresented and omitted from reporting.

TABLE I: Qualitative comparison of framework properties.

Framework Property	Ours	AnalogCoder	AnalogGenie
Simulator-Based Training	*		
Explicit Specification Targeting	*	*	
Process Technology Compliant	*		*
Netlist Syntax Enforced	*		
Expert Topology Input	*	*	*

TABLE II: Summary of comparative framework performance. * indicates values were taken from the pretrained model.

Task	Metric	Ours	AnalogCoder (GPT-3.5)	AnalogCoder (GPT-4o)	AnalogGenie
RO	Netlisting Validity (%)	100	53.3	83.3	-
	Simulation Validity (%)	90.1	0.0	26.7	-
	Specification Fulfilling (%)	13.6	0.0	0.0	-
	Novelty (%)	100	86.7	83.3	-
Comp.	Netlisting Validity (%)	100	53.3	60.0	-
	Simulation Validity (%)	99.1	30.0	46.7	-
	Specification Fulfilling (%)	99.1	26.7	43.3	-
	Novelty (%)	100	73.3	46.7	-
OTA	Netlisting Validity (%)	100	73.3	100	93.2
	Simulation Validity (%)	98.5	0.0	46.7	0.56*
	Specification Fulfilling (%)	76.5	0.0	6.7	0.25*
	Novelty (%)	100	96.7	83.3	99.0

Broadly, without simulator-based supervision during training, these models lack direct performance feedback on their outputs, making synthesis for complex tasks challenging.

Novelty: All frameworks achieved consistently high novelty rates. AnalogCoder achieved an average novelty rate of 71.1%, demonstrating the expressivity of LLMs at code-generation. AnalogGenie achieved an average novelty rate of 99%, demonstrating how expressive circuit representation can contribute to novel circuit generation. Our framework similarly attains a high average novelty rate of 100%, attributed to its fine-grained transistor-level representation.

IV. CONCLUSION

In this paper, we introduce an analog synthesis methodology enabling the discovery of new functional topologies to meet design specifications starting from known basic circuit structures. The proposed methodology combines RL with guidance from a ground-truth circuit simulator. This work further contributes several techniques towards automated analog circuit design: 1) rule-based masking to generate simulator-ready netlists, 2) generalized reward design to encourage iteration on functional designs, and 3) behavioral cloning for implicit biasing.

Our evaluation on 1) ring oscillator, 2) comparator, and 3) OTA design demonstrate the versatility of our tool on a variety of analog-mixed-signal design tasks. We yield over 90% functional designs over all tasks and demonstrate netlist validity by construction. The proposed methodology and design framework represents an important step forward towards automating analog circuit design and synthesis.

V. ACKNOWLEDGMENT

This work was supported by Intel, NSF AI4OPT, and BWRC member companies.

REFERENCES

- [1] T. Ajayi, S. Kamineni, Y. K. Cherivirala, M. Fayazi, K. Kwon, M. Sali-gane, S. Gupta, C.-H. Chen, D. Sylvester, D. Blaauw *et al.*, “An open-source framework for autonomous soc design with analog block generation,” in *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*. IEEE, 2020, pp. 141–146.
- [2] F. Guo, B. Zhou, A. Biswas, P. Kwon, Z. Liu, K. Ho, V. Stojanović, and B. Nikolić, “Bag3++: An extensible generator framework for automated layout-aware ams design,” *IEEE Open Journal of Circuits and Systems*, vol. 6, pp. 181–191, 2025.
- [3] J. Jumper, R. Evans, A. Pritzel *et al.*, “Highly accurate protein structure prediction with AlphaFold,” *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, publisher: Nature Publishing Group.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, vol. 25, 2012.
- [5] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, “Autockt: Deep reinforcement learning of analog circuit designs,” in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 490–495.
- [6] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, “Multi-objective bayesian optimization for analog/rf circuit synthesis,” in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.
- [7] K. Touloupas, N. Chouridis, and P. P. Sotiriadis, “Local bayesian optimization for analog circuit sizing,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1237–1242.
- [8] C. Chen, H. Wang, X. Song, F. Liang, K. Wu, and T. Tao, “High-dimensional bayesian optimization for analog integrated circuit sizing based on dropout and gm/id methodology,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4808–4820, 2022.
- [9] C.-C. Chang, Y. Shen, S. Fan, J. Li, S. Zhang, N. Cao, Y. Chen, and X. Zhang, “Lamagic: Language-model-based topology generation for analog integrated circuits,” *arXiv*, 2024.
- [10] Y. Lai, S. Lee, G. Chen, S. Poddar, M. Hu, D. Z. Pan, and P. Luo, “Analogcoder: Analog circuit design via training-free code generation,” *arXiv*, 2024.
- [11] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, “CktGNN: Circuit graph neural network for electronic design automation,” in *The Eleventh International Conference on Learning Representations*, 2023.
- [12] J. Gao, W. Cao, J. Yang, and X. Zhang, “Analoggenie: A generative engine for automatic discovery of analog circuit topologies,” in *The Thirteenth International Conference on Learning Representations*, 2025.
- [13] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, “Paragraph: Layout parasitics and device parameter prediction using graph neural networks,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
- [15] J. You, B. Liu, R. Ying, V. S. Pande, and J. Leskovec, “Graph convolutional policy network for goal-directed molecular graph generation,” *CoRR*, vol. abs/1806.02473, 2018.
- [16] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, “Strategies for pre-training graph neural networks,” in *International Conference on Learning Representations (ICLR)*, 2020.
- [17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017.