chosen other paper

Automated Design of Analog Circuits Using Reinforcement Learning

Keertana Settaluri, *Member, IEEE*, Zhaokai Liu, *Member, IEEE*, Rishubh Khurana, *Member, IEEE*, Arash Mirhaj, *Member, IEEE*, Rajeev Jain, *Fellow, IEEE*, and Borivoje Nikolic, *Fellow, IEEE*

***Abstract*—Analog and mixed-signal (AMS) blocks are often a crucial and time-consuming part of System-on-Chip (SoC) design, primarily due to a manual circuit and layout iterations. Existing automated solutions for selecting circuit parameters for a given target specification are often not efficient, accurate, or reliable. In order for an automated sizing tool to be practical, we posit that it must: 1) return valid results for a large range of target specifications; 2) understand where and why it is unable to meet certain specifications; 3) consider true layout parasitic simulations for complete end-to-end design; and 4) be automated, allowing most of the design effort to fall on the tool. In this article, we address these critical points by establishing an automated reinforcement learning framework, AutoCkt, by 1) successfully deploying it on a complex two-stage transimpedance amplifier and two-stage folded cascode with biasing in the 16-nm FinFet technology; 2) implementing a new combined distribution deployment algorithm to improve efficiency; 3) analyzing in-depth the efficacy of the trained agent; and 4) demonstrating the functionality of this tool when considering a topology that is highly sensitive to layout parasitics. Our algorithm not only successfully reaches unique, valid, and practical performances, but also does so in state-of-the-art run time, up to 38X more efficient than prior work. In addition, our tool averages just four parasitic simulations obtained by using the Berkeley Analog Generator, to achieve a target specification post-layout for the folded cascode. AutoCkt successfully generates LVS-passed designs with validation in process corner variation results.**

***Index Terms*—Analog sizing, Berkeley analog generator, design methodology, layout parasitics, reinforcement learning (RL).**

I. Introduction

As TECHNOLOGY nodes scale, custom circuit design becomes progressively challenging, primarily because of complex design rules and the increased prominence of layout parasitics. The end-to-end design and verification time of creating an analog circuit therefore significantly increases, placing an additional burden on expert designers who are responsible for this manual and iterative design process. Specifically, finding circuit parameters to meet a target specification relies heavily on an expert circuit designer to create and implement a design procedure and repeatedly modify the parameters in order to reach a desired specification. Each iteration, in addition, requires creating a new schematic and layout, and evaluating the circuit's performance in relevant target metrics. In order to reduce time-to-market for System on Chips (SoCs), it is crucial to identify and automate time consuming design procedures in a computation and simulation-efficient manner, without sacrificing accuracy and reliability.

Developing a practical and accurate analog sizing algorithm requires understanding the unique constraints imposed by considering layout parasitics. In particular, post-layout simulation is not only a manual process that takes significantly more time than its schematic counterpart, but also leads to large, often unpredictable variability in the simulation results. This then dictates that a sizing approach require minimal simulations to converge to the target specification, while also being robust and accurate in situations where large variations in performance exist due to parasitics. We discuss prior work in both of these contexts.

Traditionally, expert circuit designers employ a knowledge-based analog circuit synthesis methodology [1]–[4], where design considerations and parameter selection algorithms are manually codified or formulated as templates in an effort to encapsulate the designer's knowledge. This strategy, however, still requires circuit designers to be heavily involved in the sizing process, and though efficient in converging to a specification, it is not automated, and often does not consider parasitics explicitly as its effects are difficult to model. In addition it requires a large overhead in defining any new specification.

Equation-based methods [5] manually or automatically obtain constraint equations that are then optimized or solved. These solvers require very few, if any, simulation iterations, but involve manual equation formulations, do not consider parasitics, and only allow certain circuits to be characterized.

Genetic algorithms, bayesian optimization, and their optimization-based variants have also been utilized extensively, primarily via stochastic sampling to simulate a certain population of design points which are either mutated based on a fixed cost function to produce new, potentially more fit points that are simulated again in the next iteration [6], or used to generate surrogate models [7]. Genetic algorithms,

Manuscript received 20 January 2021; revised 26 April 2021 and 7 July 2021; accepted 13 August 2021. Date of publication 15 October 2021; date of current version 22 August 2022. This work was supported in part by DARPA CRAFT under Grant HR0011-16-C-0052; in part by the NSF AI Institute for Advances in Optimization under Award 2112533; in part by ADEPT; in part by the Berkeley Wireless Research Center member companies; and in part by the Qualcomm Innovation Fellowship. This article was recommended by Associate Editor S. Mohanty. *(Corresponding author: Keertana Settaluri.)*

Keertana Settaluri, Zhaokai Liu, and Borivoje Nikolic are with the Department of Electrical and Computer Engineering, University of California at Berkeley, Berkeley, CA 94710 USA (e-mail: ksettaluri6@berkeley.edu).

Rishubh Khurana is with Qualcomm Inc., Bengaluru 560037, India.

Arash Mirhaj and Rajeev Jain are with Qualcomm Inc., San Diego, CA, USA.

Digital Object Identifier 10.1109/TCAD.2021.3120547

traditionally, require many simulations to converge and are not reliable, as the converged result heavily depends on how well the initial stochastic sampling reflects the true design space. In addition, they require the algorithm to be restarted whenever a change is made to the goal. Though Bayesian optimization solutions [7] are more sample efficient, formulating a Gaussian Process model is more expensive as the dimensionality of the data increases, meaning that this solution is potentially prone to issues in scalability. Several works exist that attempt to reduce the inefficiency of these algorithms, primarily via learning [8]–[11]. Terway *et al.* [8], Wolfe and Vemuri [9], and Li *et al.* [11] do not, however, demonstrate results with layout parasitics, making it difficult to claim the reliability and practicality of these results. Hakhamaneshi *et al.* [10] demonstrated that their combined neural network boosted genetic algorithm can function on complex circuits with post-layout simulation following parasitic extraction (PEX), but the algorithm requires many PEX simulations to converge, making it difficult to scale to more complex circuits. Bayesian optimization methods [7], despite being more sample efficient, have not yet been proven with parasitic simulations.

Reinforcement learning (RL) has been applied to analog sizing as well, where an agent traverses the design space while trying to maximize a reward function. Lee and Oliehoek [12] explored initial results using this method, but only tests with NGSPICE as the simulator, which is not reflective of the true complexity of state-of-the-art processes. Wang *et al.* [13], [14] and use enhanced RL methods with recurrent or graph convolutional neural networks, respectively, that are trained using policy gradients. These algorithms do not consider layout effects and train the agent to meet only one target specification, requiring many circuit simulations to converge to other targets. Wang *et al.* [14] does however, show initial promise in using generalization to transfer information across topologies. Yang *et al.* [15] accelerated RL convergence by incorporating local constraint satisfaction while considering process corners, but fails to consider layout parasitics. Our prior work, Settaluri *et al.* [16] used model-free RL and requires minimal runtime simulations to converge with high accuracy during deployment, while showing results with layout parasitics, but has only been tested on simple amplifier designs with a small number of tunable parameters.

Despite the existence of automated layout synthesis tools [17], prior sizing algorithms opt to use parasitics models [18]–[23] to approximate layout effects without impacting simulation time. Approximate models show initial promise in predicting parasitic effects, but need to be further analyzed for accuracy and scalability on more complex circuits in newer and more parasitic-dominant technology nodes. Methodologies also exist that use lookup tables a-priori to simulate all relevant designs, but are time consuming and not scalable, as they require collecting a large number of layout simulations [24]. Garitselov *et al.* [25], [26] proposed a neural network metamodel that performs optimization in a sample efficient manner, requiring just two layout simulations to converge to a solution. Their methodology, however, requires a designer to manually create layout, and only tests on smaller-order circuits.

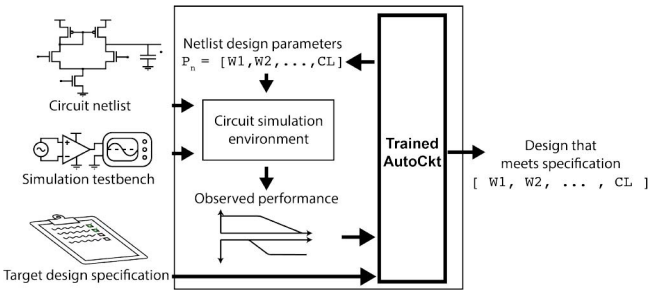

Fig. 1. Top level system diagram showing that AutoCkt takes as input a circuit netlist, its simulation testbench, and a target design specification to find parameters for the circuit such that it meets the specification.

In summary, there is need for an accurate and reliable method for solving analog circuit sizing, that not only considers the many stages of the design process that experts use to validate an analog design, from schematic, layout, and process corner simulations, but must also do so with reasonable run times. This, in addition to understanding how and why the tool works, is crucial in the adoption of any sizing framework in the analog design community.

A. Our Contributions

Inspired by the structured iterative design process practiced by expert analog designers, we propose AutoCkt (Fig. 1), a machine learning framework to automatically optimize analog circuits. In this article, we present an in-depth analysis of AutoCkt [16], and demonstrate how we scaled our tool to enable testing and validation on two complex circuit topologies in 16-nm FF, with demanding target specifications, while considering the true effects of layout parasitics without the use of approximate modeling. We show that our framework is capable of obtaining performances to meet a target specification even when there are significant differences in simulation results once layout parasitics are included, and requires an average of four post-layout extracted simulation steps to achieve a practical target specification. We also analyze where and when our algorithm performs well, in addition to assessing regions of failure for hard-to-reach targets.

This article is organized as follows: Section II introduces the RL framework and how layout simulations are incorporated in the algorithm, Section III details testing on a two-stage transimpedance amplifier with negative g_m load with a comprehensive testbench setup with layout, Section IV shows schematic, layout, and corner simulation results on a parasitic-sensitive two-stage folded-cascode amplifier, and Section V presents a summary of the analysis and results.

II. AutoCkt RL Approach

RL is a machine learning technique known to solve complex tasks in many systems. Specifically, it consists of an agent that interacts with its environment through a trial-and-error process that mimics learning in humans. It is a simulation-in-the-loop method, and has the ability to verify outputs from a true simulation source.

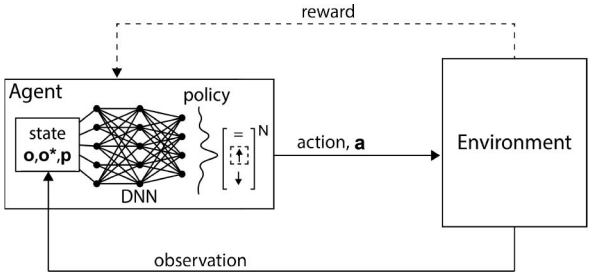

Fig. 2. RL algorithm showing that the input to the network are the current performance, target specification, and current parameters, and the output is a probabilistic distribution that is sampled to determine whether to increment, decrement, or retain each circuit parameter.

At each environment step in our algorithm, the RL agent, which contains a neural network, observes the state of the environment and takes an action sampled from the output of this probabilistic distribution. The environment then returns a new state which is used to calculate the reward for taking that action. The agent iterates through a trajectory of multiple environment steps, accumulating the rewards at each step, until the target is met or a predetermined maximum number of steps is reached. After running multiple trajectories, the neural network is updated to maximize the expected accumulated reward using policy gradients.

Let N be the number of parameters to tune in our circuit, and M be the number of different target design specifications we wish to achieve. We define our parameter space as $\mathbf{x} \in \mathbb{Z}^N$ and the target design specification space as $\mathbf{y} \in \mathbb{R}^M$, where $\mathbf{y}$ is normalized to a fixed range. The parameter space, originally continuous in $\mathbb{R}^N$, is discretized to K grids: $\{\mathbf{x} \in \mathbb{Z}^N : 0 \leq x_i < K, i = 1, \ldots, N\}$.

A. Trajectory Generation

Upon reset, the circuit parameters are initialized to a fixed center point $(K/2)$. The neural network then uses the observed performance $\mathbf{o}$ (created by simulating the circuit) and target specification $\mathbf{o}^*$, as well as current parameters $\mathbf{p}$ to output a probabilistic distribution that is sampled to determine whether to increment, decrement, or retain the same value for each circuit parameter (shown in Fig. 2). The agent has H total simulation steps to reach $\mathbf{o}^*$, where H is nominally set to K. If the objective is reached before H steps, the trajectory ends. We note that the fixed reset point is chosen strategically to ensure the agent reaches all K points in the parameter space within the allocated H steps.

B. Training With Schematic Simulations

To train the RL agent, we select M different target specifications to allow the agent to understand many different regions of the design space. Specifically, these M targets are chosen by randomly sampling each metric in the specification between a predefined minimum and maximum range set by the user, and dictated the circuit topology and application. L trajectories are generated, whose targets are chosen from M. The reward for each trajectory is obtained by accumulating the rewards for every action, formulated as the unweighted sum of the bounded normalized differences between each schematic-simulated metric performance, o, and its target, o^*, multiplied by a scaling constant, β

$$r_o = \min\left(\beta \frac{o - o^*}{o + o^*}, 0\right) \tag{1}$$

$$r = \sum_{o \in \mathbf{o}} r_o. \tag{2}$$

β is 1 for metrics being maximized, -1 for metrics being minimized, and $-1 < \beta < 0$ is for minimized metrics that are not generally defined as hard constraints, such as power. Thus, more negative rewards are given for being further below target metrics being maximized or further above target metrics being minimized. The agent is not rewarded for over satisfying any one given metric; r_o is clipped to zero for such cases.

If the sum of all r_o reward components is zero, the obtained performance meets the target specification, and the agent receives an additional positive scalar term of 10. In our studies, this scalar term allows the algorithm to train faster, as it gets a distinct and higher continuous R in cases where it meets the target. We note that this broad reward formulation allows for the same function to be used in the variety of different amplifiers we have tested, without the need for tuning to specific circuit topologies

$$R = \begin{cases} r, & \text{if } r < 0 \\ 10 + r, & \text{if } r = 0. \end{cases} \tag{3}$$

To make training more sample efficient, we utilize the sequential sizing methodology employed by expert designers. In particular, once an initial guess about parameters is made, designers first run DC operating point to ensure transistors are in the expected region of operation. After this verification, AC simulation is used to obtain gain, unity-gain bandwidth, phase margin, and power values. Once these are again verified, transient, and noise simulations are obtained for the final metrics.

By following these steps, designers gain intuition about how valid each stage of the simulation process is, reducing overall runtime. For instance, when we randomly select parameters for a folded-cascode circuit, 85%of the designs do not pass DC operating point. Since the average runtime for the entire testbench is 43 s, designers save time in the sizing process by not having to run needless simulations.

In our algorithm, if at any point a testbench is invalid, subsequent testbenches are not run and the target metrics from those simulations are set to -1, which is the smallest negative number in the bounded reward function in (1). In addition to reducing simulation time, this technique also breaks down a multistep optimization problem to easier subgoals, allowing the agent to learn quicker. We note that these validity tests are agnostic to the specific circuit type or topology, meaning that this methodology can be ported to many different designs. Our algorithm also functions without this technique, but could take longer to train.

To train the RL agent, a batch of trajectories is collected with the schematic simulator, and the rewards, states (comprised of $\mathbf{o}$, $\mathbf{o}^*$, and $\mathbf{p}$) and actions (increment, decrement or retain each parameter) are used to update the weights of the

Algorithm 1: Sparse Subsampling Training Algorithm

```
Initialize O* ← R^(MxN)                    #create training targets
env ← schematic simulator
while Σ R/L < 0 do
    Dataset ← []
    R ← []
    for len(batchsize) do
        o* ← O*                             #select one target
        t = Collect_Trajectory(o*, horizon length, env)
        Dataset ← append(Dataset, t)
        R ← append(R, reward(t))
        PPO_Update_NN_Weights(R,Dataset)
    end
end
```

neural network with gradient descent. The neural network has three layers with 50 neurons each, and is fixed across the different circuit topologies. Proximal policy optimization (PPO) is a policy gradient method that finds a precise local optima as opposed to other approaches like simulated annealing, which are approximate, and is used to formulate the objective function that maximizes the average accumulated reward. PPO is known to be stable and efficient during training by controlling how large the policy updates are at each iteration.

Training sessions are conducted several times to ensure that AutoCkt is robust to variations in a random seed. Training ends when the mean reward is greater than or equal to 0, signifying that the agent reaches the training target specifications for most trajectories. This training process is summarized in Algorithm 1. We use OpenAI Gym and Ray, a framework [27] for running distributed RL tasks, to efficiently implement our algorithm across a many-core CPU while also minimizing horizon length.

C. Deployment With Layout Simulations

During deployment, the trained agent generates trajectories with unique target specifications, different from those in training. Our modified generalization algorithm uses the Berkeley Analog Generator [28] to consider layout parasitics.

BAG allows designers to create, verify, and use process-portable circuit generators. The framework requires a template schematic and a manually scripted layout generator, both of which read in parameter sizes, component values, and other tunable parameters to determine how to generate the circuit. Once this schematic and layout are generated, designers can create testbench scripts that automatically synthesize schematic and PEX simulation results from ADE-XL.

The generalization algorithm is shown in Fig. 3. Instead of placing the trained agent in an environment that solely consists of parasitic simulations [16], we combine schematic and parasitic distributions by taking steps from both environments. This not only leads to faster runtime, as it avoids needless and costly parasitic simulations, it is also more robust to this variation. This robustness is crucial, as one of the circuit topologies we select is highly sensitive to parasitics.

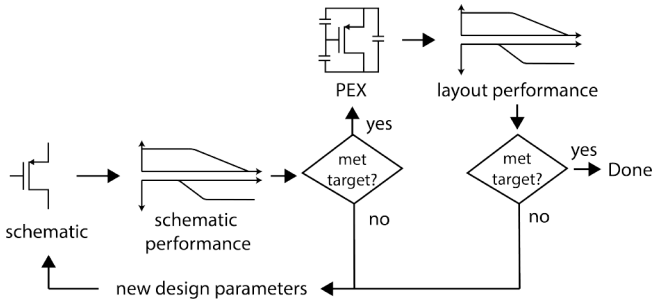

Fig. 3. Parasitic deployment algorithm that shows how schematic and layout simulation distributions are combined to obtain a target specification.

Algorithm 2: Deployment Algorithm Considering Layout Parasitics

```
o* ← R^M
env_1 ← schematic simulator
env_2 ← parasitic simulator
while True do
    o, p ← step(o, o*, p, env_1)
    schematic_reward = calc_reward (o, o*)
    if schematic_reward ≥ 0.0 then
        o, p ← step(o, o*, p, env_2)
        parasitic_reward = calc_reward (o,o*)
        if parasitic_reward ≥ 0.0 then
            return p, o, (reached ← True)

    if step_count > H then
        return p, o, (reached ← False)
end
```

Upon receiving a target specification, the trained agent starts by running schematic simulations until it reaches $\mathbf{o}^*$. Instead of receiving a positive reward for that step for obtaining a valid solution in schematic, we simulate the same parameters the agent used to meet this target, in an environment that considers layout parasitics. If the agent does not meet the target once parasitics are considered, the layout performance is given back to the agent, which then uses this information to adjust in schematic until it reaches another set of viable parameters which are used to run PEX. This process continues until the agent meets the target when parasitics are considered, or has reached the maximum allocated steps H, shown in Algorithm 2. This combination of schematic and layout simulation deployment allows the agent to traverse a distribution it knows, in order to better understand and generalize to a distribution it has not seen before, allowing it to be more successful in reaching a target.

III. Two-Stage Transimpedance Amplifier With Negative Gm Load

Previously, we demonstrated AutoCkt's successful functionality on a transimpedance amplifier, two-stage amplifier, and two-stage transimpedance amplifier with negative g_m load [16]. In this article, we extend the results on the two-stage

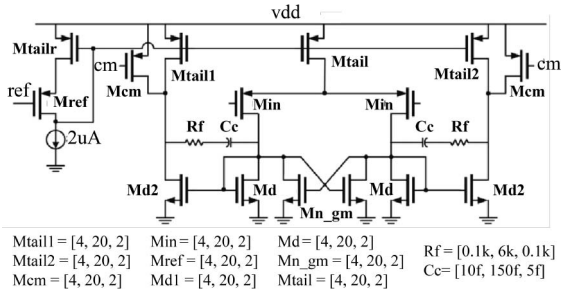

Fig. 4. Two-stage transimpedance amplifier schematic.

TABLE I
PERFORMANCE RANGE SAMPLED DURING TRAINING FOR OPERATIONAL TRANSIMPEDANCE AMPLIFIER

Metric	Minimum	Maximum
Gain (dB)	35	75
Unity gain frequency (GHz)	0.01	1.0
Phase margin ($^\circ$)	60	75
Settling time (ns)	10	75
Input referred noise (μV_{rms})	1100	1900
Differential voltage swing (V)	0.5	0.7
Dissipated power (μA)	1	100

TABLE II
OBTAINED SCHEMATIC PERFORMANCES OF TRAINED AGENT FOR OPERATIONAL TRANSIMPEDANCE AMPLIFIER

Metric	Minimum	Mean	Maximum
Gain (dB)	42.0	48.3	75.5
Unity gain frequency (GHz)	0.05	0.08	0.26
Phase margin ($^\circ$)	60.2	64.2	76.7
Settling time (ns)	6.9	34	49.8
Input referred noise (μV_{rms})	971.6	1938.3	3198.1
Differential voltage swing (V)	0.48	0.63	0.67
Dissipated power (μA)	12	32	55

transimpedance amplifier with negative g_m load in the TSMC 16-nm finFet technology to incorporate more target metrics, as well as reflect the modified training and deployment algorithm presented in Section II. Fig. 4 shows the circuit topology and discrete tunable MOS, resistor, and capacitor parameters with their ranges [min, max, increment]. The target metrics are open-loop gain (A_{vo}), phase margin (pm), unity gain frequency (f_t), settling time (t_{set}), dissipated power (P), input-referred noise (V_{noise}), and differential output voltage swing, (V_{pp}), with a selected load capacitance of 0.22 pF. The target specifications are selected from the range in Table I, by sampling regions around an expert performance.

A. Training Results

The minimum, mean, and maximum reward per trajectory for each iteration of training (represented as a number of simulations or steps) is used to assess whether the algorithm trained successfully with schematic simulations. Fig. 5 shows that the mean reward reaches zero after training has completed, meaning that the agent, on average, has learned to reach many training target specifications.

We test how well our algorithm understands the design space by giving the trained agent new and previously unseen target specifications. Table II shows the agent obtained 393 unique points with a wide range of schematic performances

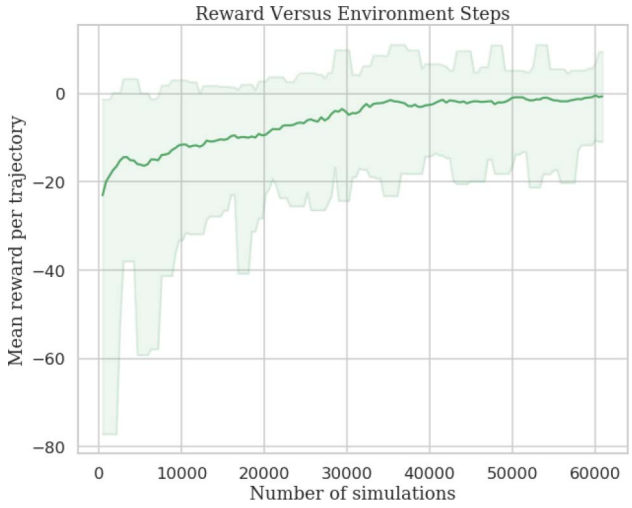

Fig. 5. Mean reward per trajectory for the two-stage transimpedance amplifier.

TABLE III
LAYOUT PERFORMANCE OBTAINED BY AGENT COMPARED TO EXPERT

Metric	Expert	Agent
Gain (dB)	75	76.6
Unity gain frequency (GHz)	0.16	0.17
Phase margin ($^\circ$)	69	65
Settling time (ns)	10	9.7
Input referred noise (μV_{rms})	1400	1358
Differential voltage swing (V)	0.57	0.59
Dissipated power (μA)	54	52

TABLE IV
ROUNDED AVERAGE RSC FOR TRANSIMPEDANCE TWO-STAGE AMPLIFIER IN SCHEMATIC AND LAYOUT FOR 60 PERFORMANCES

Seed	Schematic RSC			Layout RSC
	Min	Mean	Max	
1	2	20	45	1
2	1	34	40	1
3	1	28	40	1

in each metric. In addition, the agent on average reaches these performances in 20 simulation steps.

The trained agent is deployed with layout and schematic simulations using the algorithm from Section II. We compare the results to an expert design in Table III and show that our algorithm is able to out-meet the expert on every metric except phase margin, where there is a 5% degradation, with 1 layout and 17 schematic simulations. This required simulation count (RSC) is summarized in Table IV for the 60 unique tested layout performances the agent obtained.

Compared to our own prior deployment algorithm [16] which required 23 layout simulations to converge, our proposed approach uses 22.3× fewer runtime simulations. We note that this topology has significant degradation in schematic versus layout simulation results, with up to 28% difference between the metrics. The difference between schematic and PEX simulations for each metric, however, is mostly linear, meaning that our algorithm (and a potential layout parasitics model) could predict this degradation. In the next section, we introduce a more parasitic-sensitive topology to demonstrate our algorithm's robustness in cases where schematic and

TABLE V
REGIONS OF INTEREST IN PERFORMANCE FOR FOLDED CASCODE IN PIPELINE ADC APPLICATION

Metric	Target Perf. 8-bit	Target Perf. 10-bit
Gain	$> 54dB$	$> 60dB$
Unity gain frequency	$> 980MHz$	$> 1.20GHz$
Phase margin	$> 60°$	$> 60°$
Settling time	$< 1.7ns$	$< 1.7ns$
Input referred noise	$< 750\mu V_{rms} \pm 40\%$	$< 380\mu V_{rms} \pm 40\%$
Differential voltage swing	$> 0.8V$	$> 0.8V$
Dissipated power	$< 500\mu A$	$< 600\mu A$

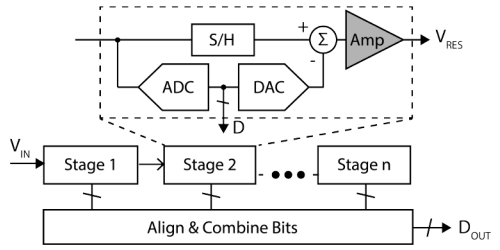

Fig. 6. Pipeline ADC block diagram with amplifier circuit in gray.

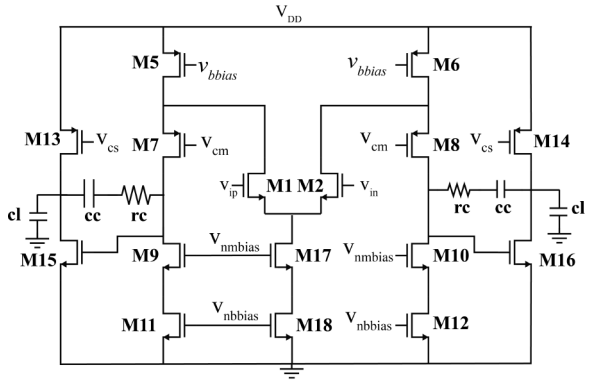

Fig. 7. Folded-cascode core topology: 22 parameters total.

layout simulation results are significantly nonlinear and more difficult to model.

IV. FOLDED-CASCODE AMPLIFIER

We select a complete fully differential two-stage folded-cascode amplifier with biasing circuitry to evaluate AutoCkt's performance on a more complex, parasitic-sensitive topology. We adapt the folded cascode from [29] to enable functionality in a 16-nm finFet technology, and choose target specifications (Table V) to match practical requirements in a pipeline analog-to-digital converter (ADC), whose block diagram is shown in Fig. 6. The folded-cascode consists of biasing and amplifier stages, with ideal common-mode feedback, illustrated in Figs. 7 and 8. In total, the circuit has 42 parameters, including transistor sizes, load and compensation capacitors and resistors, and a bias current source. This circuit not only has a complex design space, but also shows significant behavioral changes once layout parasitics are incorporated, making it an ideal candidate to test the robustness of our RL methodology.

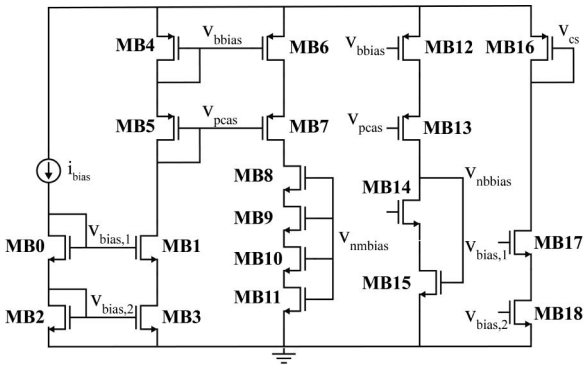

Fig. 8. Biasing topology: 20 parameters total.

TABLE VI
TARGET SPECIFICATION RANGE SAMPLED FROM DURING TRAINING FOR FOLDED CASCODE

Metric	Minimum	Maximum
Gain	$50dB$	$65dB$
Unity gain bandwidth	$1000MHz$	$40GHz$
Phase margin	$> 60°$	$100°$
Settling time	$0.5ns$	$100ns$
Input referred noise	$300\mu V_{rms}$	$600\mu V_{rms}$
Differential voltage swing	$0.6V$	$0.8V$
Dissipated power	$600\mu A$	$800\mu A$

We use the same performance metrics as the two-stage transimpedance amplifier: A_{vo}, pm, f_t, t_{set}, P, V_{noise}, and V_{pp}, and select an 8 and 10 bit pipeline ADC application for this topology, whose derived target specifications are summarized in Table V. In the results, we discuss two cases: one where the C_L given to the agent ranges from 0.1–10 pF, supporting established equation-based analyses, and the other where C_L varies from 8–30 fF, giving insight into how this agent functions in an atypical design case.

A. Schematic Training Setup

To train the two-stage folded-cascode design, we first determine the range from which the target specifications are randomly subsampled during training, and what parameter constraints are enforced for this folded-cascode topology.

1) Target Specification Selection: The selection of target specifications is chosen such that it encompasses a wide range of values, and includes the regions of interest from Table V. Providing regions above and below the reference allows the algorithm to learn the relationship between parameters and metric in a variety of different operating ranges. This aspect of training allows the agent to perform well once layout parasitics are concerned, as the performance degradation might require the agent to size for a different region in the design space. The selected range, shown in Table VI, is then used to randomly sample from and generate the objectives $\mathbf{o}^*$ that are used to train the agent. In our experiments, we show the affect of varying the number of $\mathbf{o}^*$s chosen.

2) Parameter Selection: The tunable parameters for this circuit topology include length and number of fingers for each MOS device, feedback resistance, feedback and load capacitances, multiplier, and the input current source. Each

TABLE VII
PARAMETER CONSTRAINT SELECTION

Employed Constraint	Parameters	Design Space Size
None	42	$5.5e43$
Matching	32	$2.1e33$
Matching + Biasing	29	$1.6e30$

parameter range is derived from the BAG layout generator used to consider layout parasitics, and has 11 unique values to select from. We focus on uniform channel length devices when parasitics are included, limited by the simplicity of the BAG layout generator, however.

To understand the impact of constraining the parameter space, we take two extremes: one where every parameter in the circuit is treated as a separate variable with unique tunable ranges (in the folded-cascode case, this would amount to 42 variables), and the other where design equations are incorporated such that only a finite number of variables exists for the agent to tune. On one hand, the large parameter space allows the agent flexibility in determining potential sizing solutions, but at the risk of not following certain rules that circuit designers know must hold for valid behavior (i.e., matching). On the other hand, incorporating design insight allows the algorithm to train faster, but also introduces bias, potentially reducing the agent's explorative power. In an automated analog circuit design tool this tradeoff is extremely important, as the goal is to not only return a valid solution, but also provide the tool flexibility in finding unique solutions.

Thus, in our experiments we provide three different parameter versions, one where the agent receives no outside constraint assistance, one where the agent receives basic necessary constraints, such as matching, and the last where the agent receives matching and easily coded yet still limited constraints, such as certain biasing circuitry transistors having widths and lengths multiples of one another. These three versions are summarized in Table VII. We note that no test involves explicitly formulating design equations, as that prevents the tool from being automated.

B. Schematic Training and Deployment Results for Folded Cascode

In this section, we show the results of training and deploying the agent on the two-stage folded cascode, as well as additional analyses to understand the effect of varying targets, constraints, and performance.

1) Varying Target Specifications During Training: The mean episode reward per number of steps is depicted in Fig. 9, and shows that the agent successfully reaches the stopping condition by obtaining many target specifications.

Fig. 9 also shows the effect of varying the total number of training target specifications given to the agent to meet. It shows that the number of simulations required to train is inversely correlated to the number of targets given to the agent, with more targets helping the agent train faster. This, however, is not significantly different compared to the total number of simulations, with all three experiments falling within 12 000 simulation steps of each other. These results are expected, as

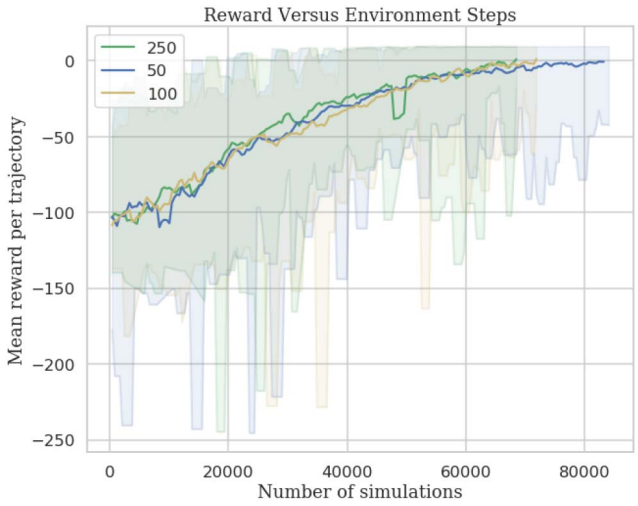

Fig. 9. Mean reward per trajectory for 50, 100, and 250 training target specifications.

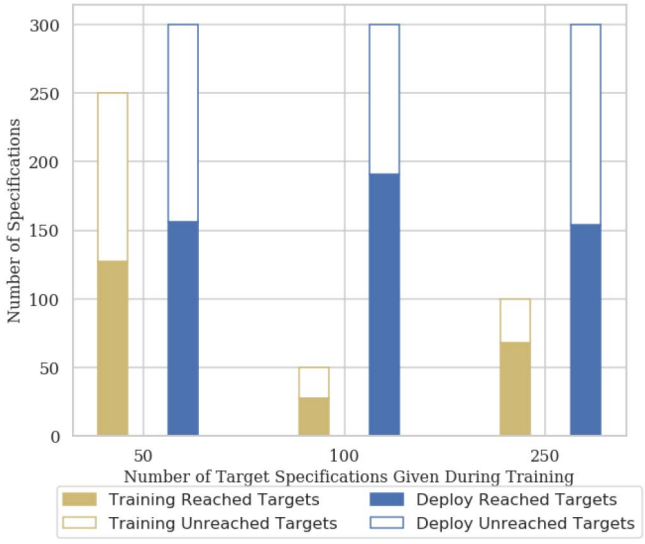

Fig. 10. Deploying trained agent on target specifications not seen before.

the agent is able to randomly select targets for each trajectory at every training iteration. With a higher number of targets, the agent has a larger variation with this selection, meaning that a particular sample of targets could help the agent reach the stopping condition faster. We do note that there appears to be a larger step-to-step variation due to this.

To understand intuitively what Fig. 9 means in the context of the circuit sizing problem, the number of target specifications met by the agent during training is calculated. The results are summarized in Fig. 10, which also shows the effect of varying the number of training target specifications. The agent on average meets just over half of the targets across the three different experiments. As we will show in our failure analyses in Section IV-C), this 55% success rate does not mean the agent fails at learning the relationship between parameters and performance for this topology. Instead, we demonstrate the targets we have chosen are pushing performance boundaries for this parameter range and technology.

2) Varying Number of Constraints: Fig. 11 shows the results of varying the number of constraints applied to the

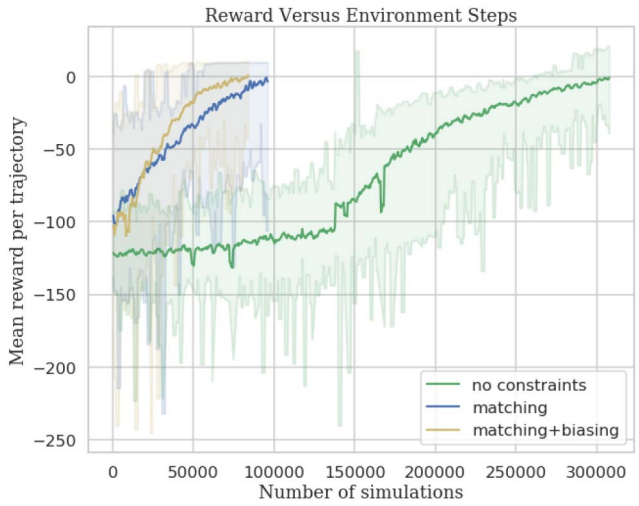

Fig. 11. Mean reward per trajectory for varying number of constraints.

TABLE VIII
COMPARISON OF PRIOR ANALOG SIZING METHODS WITH RSC TO CONVERGE AND NUMBER OF SPECIFICATIONS REACHED VERSUS TESTED

Prior Work	RSC	# Specs Reached / # Specs Tested
GA + NN [10]	494	4/50
GA [30]	51973	1/1*
AutoCkt (training)	71001	N/A
AutoCkt (deploy)	13	191/300

*Only one target specification tested on vanilla genetic algorithm because of RSC

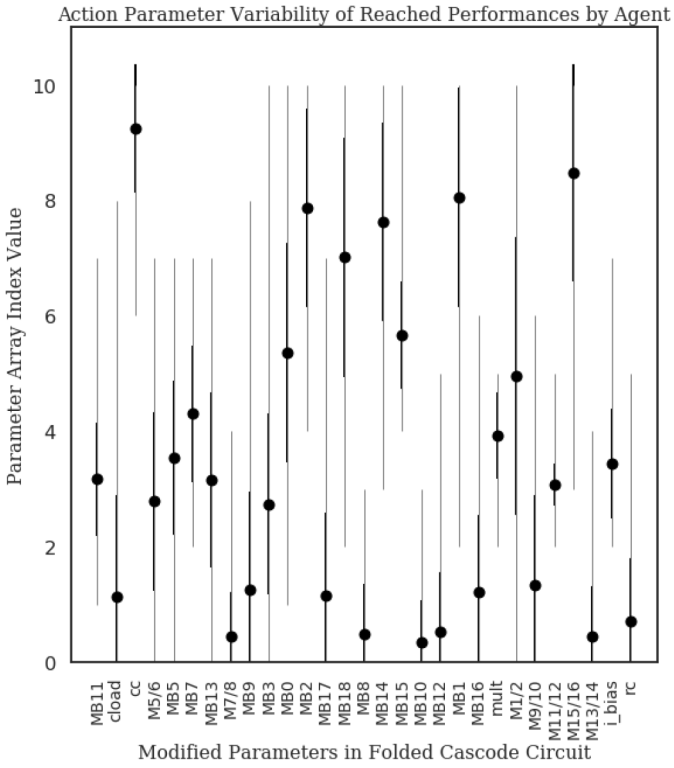

Fig. 12. Parameter variation for reached performances.

agent during training. This plot shows that as the number of parameters increases, the longer it takes the agent to train. In all cases, the agent is able to successfully maintain an upward increase in mean reward and reach the stopping condition. The no constraints model, however, requires considerably more simulation steps to converge. This is likely due to difficulty in obtaining feasible design regions when the parameter count is very high. In our studies, we have found that randomly selecting parameters in the space has an 85% chance of not passing DC simulation constraints, which could explain the initial flat slope for this experiment.

3) Deployment With Schematic-Only Simulator: We deploy the trained agent on 300 new target specifications it has never seen before, in the same range as the randomly sampled targets during training in schematic only, and summarize the results in Fig. 10. The percentage of reached versus unreached targets is similar to the results of training, with approximately 50% of them being successful. This in turn means that the agent does not overfit on the targets given to it during training.

In addition, the trained agent requires on average just 12.5 simulation steps to converge to a set of parameters that meets a given target specification. We compare this RSC with prior state-of-the-art, which include genetic algorithms and their variants, in Table VIII. To generate the best possible comparison, the hyperparameters for each of these approaches is optimally selected based on the fastest convergence on a test target specification. These hyperparameters are then fixed across the rest of the target specifications. We note that for the genetic algorithm boosted with neural network [10], we do not tune the neural network's hyperparameters, as these parameters are not modified in our RL framework. Table VIII also includes the RSC to train the RL agent, which is considerably higher than prior work. This training cost is amortized, however, in two ways: 1) reusing the trained agent on other target specifications for the same circuit topology leads to a runtime convergence that is up to 38× faster than prior work and 2) deploying the trained agent on layout simulations reduces post-layout runtime convergence, as we will show in Section IV-D.

C. Reachability Analyses in Schematic Training for Folded Cascode

This section analyzes performances that AutoCkt obtains in order to posit that: 1) our framework reaches variable, valid, performances and parameters in the design space and 2) failure in reaching a target specification points to unreachability of these targets as opposed to failure of the algorithm.

1) Variability and Validity of Reached Performances and Parameters: Two aspects are important in assessing whether the agent understands the design space and tradeoffs for this folded-cascode topology: 1) how many unique parameters and 2) how many unique performances the agent obtains in order to meet the deployment target specifications. In other words, we would like to ensure that the agent does not converge to one set of parameters with the highest performance to satisfy all of the targets given to it.

The minimum, maximum (at most 11), mean, and standard deviation of the array index values for each tunable parameter in the folded cascode are shown at each end of the gray line in Fig. 12, the black circle, and the black line, respectively, for

TABLE IX
REACHED PERFORMANCE RANGES AND ONE SAMPLE DESIGN OBTAINED BY AGENT COMPARED TO EXPERT ($100o^*$)

Metric	Min	Mean	Max	Agent	Expert
UGBW (GHz)	2.67	7.84	14.8	10.5	10.0
Gain (dB)	56.7	63.2	65.7	49.4	66.1
Noise (μV_{rms})	303.4	380.4	531.1	352	352
Power (μA)	370.1	613.0	803.1	1.24	3.33
PM ($^\circ$)	59.6	73.4	93.7	66.3	85.2
Tset (ns)	1.2	3.6	10.5	0.89	1.7
Vswing (V)	0.62	0.86	0.97	0.83	0.79
C_L(F)	8f	20f	30f	1p	1p
Efficiency ($MHzpF/mA$)	-	-	-	8500	3000

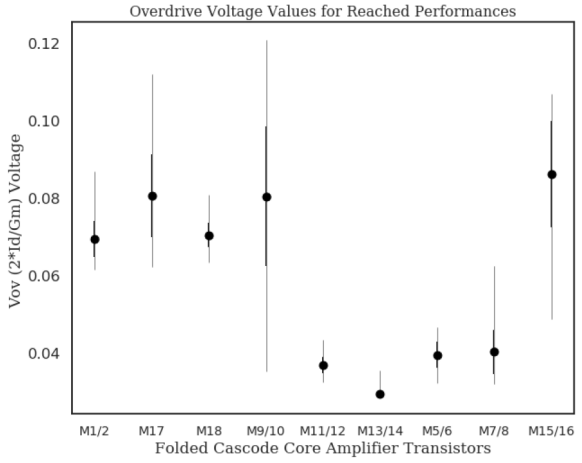

Fig. 13. Overdrive voltage (V_{ov}) analysis for reached performances.

TABLE X
REACHED PERFORMANCE METRIC DISTRIBUTION FOR UNREACHED TARGETS

Number of Metrics Met	Percent of Total Performances
3	0.85%
4	9.4%
5	43.6%
6	46.1%

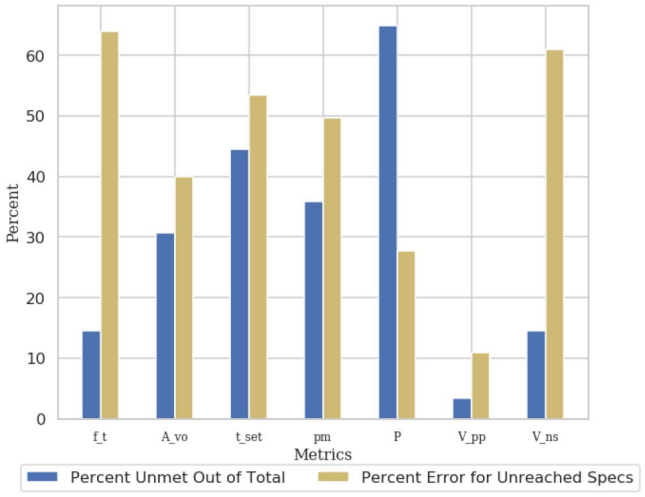

Fig. 14. Metric failure percentage and percent difference to target for those failed metrics, where *f_t* is unity gain frequency, *A_vo* is gain, *t_set* is settling time, *pm* is phase margin, *P* is power, *V_pp* is voltage swing, and *V_ns* is noise.

designs the agent obtains to successfully reach a target specification. The variability in parameter values clearly shows that the agent understands and reaches performances with diverse exploration over the range of parameters.

The minimum, mean, and maximum obtained performances are summarized in Table IX, showing that the agent reaches a diverse range of performances. In addition, the maximum efficiency, calculated by $f_t C_L / i_{\text{bias}}$ is greater for the agent-converged design than that of the expert, validating that for the higher load capacitance case, the designs are comparable.

To analyze the validity of the agent's designs from a circuit's perspective, we examine transistor operating points to ensure that devices operate in the correct regime. For this topology, the overdrive voltage values for each of the reached performances are calculated and averaged. The results are summarized in Fig. 13 and show that our framework finds reasonable and valid operating regions for each amplifier transistor.

2) Analysis on Unreached Target Specifications: The distribution of unreached target specifications is analyzed according to the performance with the highest reward less than zero that the agent converges to. In such cases, we look at the number of target metrics met, and summarize the results in Table X. This table shows that 89.7% of the unreached targets consist of greater than five metrics being met, out of a total of seven. This means that the agent optimizes and obtains performances that are close to the target. Fig. 14 further shows the unmet target specifications, categorized by what percent of metrics are not met. From this plot, it is clear that settling time, phase margin, and gain constraints are a significant portion of the unmet distribution (power is ignored as it is not a hard constraint). The percentage error between the obtained metric performance and its associated target is also shown in the figure. We note large errors in unity gain frequency, settling time, and voltage swing, but posit that this is due to the selected targets pushing the performance boundary for this circuit topology, as we will analytically show in the next section.

3) Distribution of Unreached Target Specifications: We analyze the distribution of unreached targets, and group each metric based on whether they are within 15% of the maximal (for metrics being maximized) or minimal range (for metrics being minimized) initially used to generate the targets during training, with the exception of phase margin which only has a 5% range to account for the smaller variation. The results are summarized in Fig. 15. This histogram shows that 98.6% of distribution of target specifications the agent does not meet stems from at least one metric nearing its upper (maximized metrics) or lower bound (minimized metrics). In addition, Fig. 15 also plots the unmet target metrics over the total number of targets classified within each bound, which increases with more metrics being in the toughest bound. In addition, because these target metrics are randomly sampled from a fixed range, the probability of sampling six or seven metrics within the toughest bound is very low, explaining why fewer targets exist in these categories.

Fig. 16 plots a heatmap of each of the metric tradeoffs that contribute to the histogram from Fig. 15. The highest number

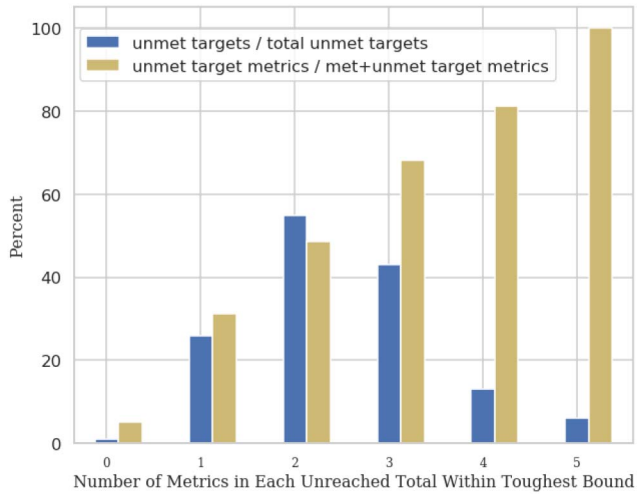

Fig. 15. Distribution of unreached targets binned by number of metrics that are within their toughest bound.

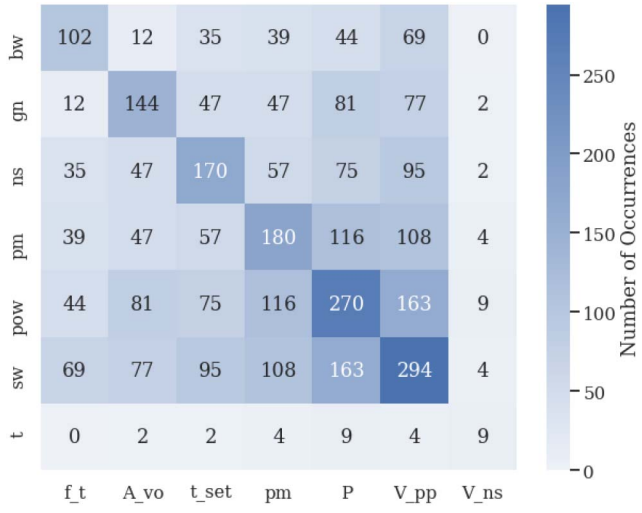

Fig. 16. Heatmap of number of occurrences of different metric tradeoffs in unreached target specifications, where *f_t* is unity gain frequency, *A_vo* is gain, *t_set* is settling time, *pm* is phase margin, *P* is power, *V_pp* is voltage swing, and *V_ns* is noise.

of tradeoffs occurs between output voltage swing and the other metrics, including power and phase margin. From a circuits perspective, this is intuitive as voltage swing determines the bounds of other metrics. In addition, gain and power, gain and swing, and noise and power are featured as well. It is apparent that these common circuit tradeoffs are being considered by the agent during the training process.

In summary, regions where the agent does not meet target specifications are posited to be unreachable due to a significant percentage of these targets having metrics that fall in the toughest bound, with metric-to-metric tradeoffs further proving that these tradeoffs are common when designing circuits.

D. Generalization to Layout Parasitics

Prior analog sizing tools either lack the ability to consider post-layout extracted (PEX) simulations due to RSC inefficiency and lack of automated generation of layout, or consider them via approximated parasitic models [18]–[20], which can prove inaccurate in cases where there is significant unpredictability and performance degradation between schematic and layout simulations. Sizing frameworks must consider layout and demonstrate its functionality on true circuit parasitics to be practical. In this section, we discuss how BAG is utilized to generate layouts for the folded cascode, analyze why a combination of schematic and PEX simulations is the most successful in reaching a target specification, and show how our trained RL agent performs with layout parasitics.

1) Two-Stage Folded Cascode BAG Generator: The two-stage folded cascode is designed by two subgenerators that encompass a wide range of transistor sizes: 1) the core amplifier and 2) the biasing circuitry block. The amplifier core assumes the transistors for each differential side are symmetrical and matched, and are grouped into two parts for ease of layout generation: the first consists of the tail current, input pair, and the *p*-type cascode transistors and the second contains the *n*-type and output stage transistors. This grouping allows consistent generation of LVS-passed layouts. Dummy transistors are added in empty regions to guarantee good matching.

The bias circuit generator from Fig. 8 has three sub-blocks: 1) the *p*-type cascode; 2) *n*-type cascode; and 3) tail current biasing. These sub-blocks are connected vertically allowing for easier modification of sizes. In addition, they can be adjusted without a complicated routing algorithm. Both amplifier core and bias circuit have supply routing at two sides. Connections between these blocks use a higher metal layer.

Once the BAG generator creates the schematic, layout and testbench, post-layout simulation is run on the circuit. This process takes on average 60× more time than a schematic simulation, ranging from 10 min if the sizes of transistors are small to almost 40 if they are large. This means that simply taking prior work and running with layout simulations is not feasible for more complex topologies.

2) Combined Schematic and Layout Deployment Methodology: In order to understand why a combination of schematic and layout simulations is the most successful in reaching a given target specification, we analyze the distributional difference between these datasets. Fig. 17 shows schematic and PEX simulation results for the same set of parameters. These plots show that layout parasitics affect each metric differently, with some metrics, including phase margin, settling time, and gain having distinct nonlinear patterns. In particular, only 10% of the points meet the phase margin requirement of 60° in layout. Gain also contains a sharp dropoff, attributed to self-loading, that degrades performance significantly compared to its corresponding schematic design. These features affirm that layout effects are significant for this topology, and by using a simple linear or neural network model to predict parasitics allows for optimization in a very limited range, restricting the applicability to lower performance, nonparasitic dominant circuits. The principal component analysis is conducted in Fig. 17 to show that these datasets form separate, almost nonoverlapping clusters in the design space, with parasitic results having less spread and a different centroid location. This reinforces that simply

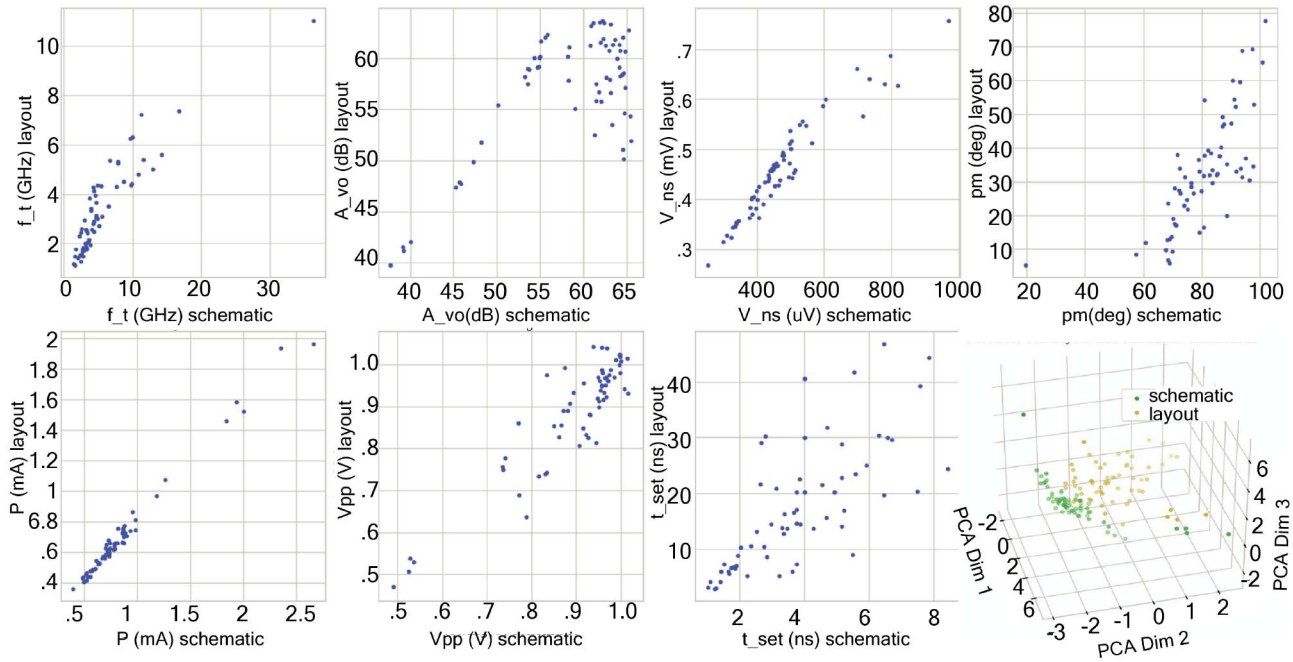

Fig. 17. Scatterplot showing the difference between schematic and layout simulation results for the folded-cascode circuit.

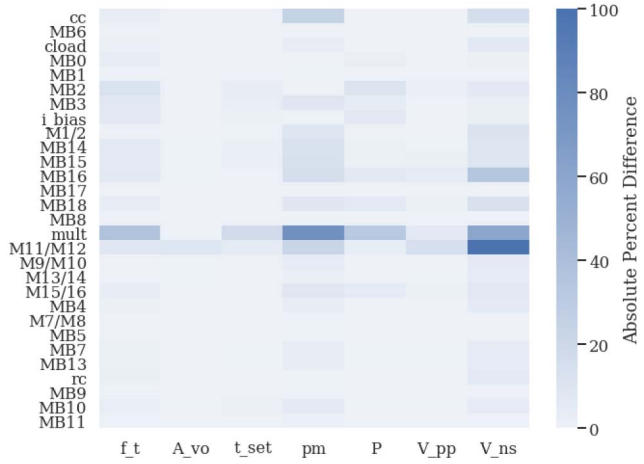

Fig. 18. Sensitivity heatmap that shows the absolute percent change in layout metric performance when a selected circuit parameters are individually modified in the design.

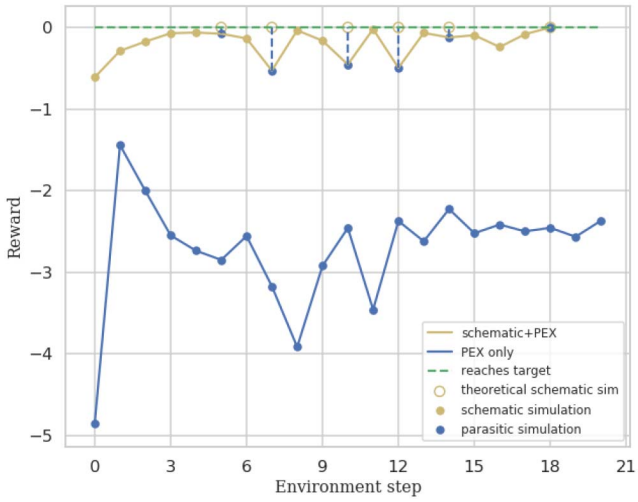

Fig. 19. Trajectory results when agent is run with a combination of schematic and layout simulations, versus only layout simulations.

taking parameters for a given performance that satisfy a target specification in schematic and translating it to parasitics does not yield reliable results. Additionally, the sensitivity of every tunable parameter to each target metric is shown in Fig. 18, by calculating the change in performance when a single parameter is modified. We find that the multiplier, nMOS load transistors, and certain bias transistors cause significant changes to the metric performances, further illustrating that self-loading is a significant factor in performance degradation in layout.

TABLE XI
REACHED PERFORMANCE RANGES WITH LAYOUT PARASITICS

Metric	Minimum	Mean	Maximum
UGBW	2.66 GHz	3.50 GHz	7.6 GHz
Gain	55.5dB	59.14dB	61.6dB
Noise	463.6 μV	543.9 μV	618.28 μV
Power	470.3μA	548.8μ A	643.4 mA
PM	60.1°	67.6°	80.3°
Tset	0.88ns	1.6 ns	3.4 ns
Swing	0.76 V	0.87 V	0.98 V

Two trajectories are plotted in Fig. 19, showing where the deployed agent only runs layout simulations [16], as well as our proposed schematic and layout simulation approach. The results show that the layout simulation approach does not converge to a specification that meets the target, shown as the green horizontal line (scaled to 0.0), despite improving the reward over the course of the trajectory. Our approach (in yellow), allows the agent to traverse the schematic simulation space, routinely checking layout parasitic simulation results and modifying the parameters in a design space that it better

TABLE XII
LAYOUT PERFORMANCES OBTAINED BY AGENT, AND TWO PRIOR WORKS FOR 8-BIT AND 10-BIT PIPELINE ADC SPECIFICATIONS

Metric	8-bit	This Work	[10]	[30]	10-bit	This Work	[10]	[30]
UGBW (GHz)	> 0.98	3.0	4.46	5.8	> 1.2	4.18	4.9	5.8
Gain (dB)	> 54	59.0	44.0	57.6	> 60	61.6	53.5	61.9
Noise (μV_{rms})	$< 750 \pm 40\%$	585.7	421.1	414.7	$< 380 \pm 40\%$	519	390.0	423.3
Power (μA)	minimize	492	501	450	minimize	589	556	581
PM ($^\circ$)	> 60	71.1	23.9	13.1	> 60	70.4	28.7	22.2
Tset (ns)	1.7	1.44	7.23	15.7	1.7	1.37	7.1	10.4
Swing (V)	0.80	0.88	0.98	0.62	0.8	0.89	0.80	0.83

TABLE XIII
FF, SS, SF, SS CORNER SIMULATION RESULTS SHOWING MINIMUM, MEAN, AND MAXIMUM OBTAINED METRIC PERFORMANCES ACROSS 60 AGENT-OBTAINED DESIGNS

Metric	**FF**			**FS**			**SF**			**SS**		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
UGBW (GHz)	2.7	3.6	5.1	X	3.3	4.8	2.75	3.6	5.0	X	3.3	4.6
Gain (dB)	53.7	56.6	58.6	X	54.3	60.1	55.8	59.5	62.1	X	53.7	61.6
Noise (μV_{rms})	509	538	702	452	525	691	495	580	681	467	514	619
Power (A)	457μ	598μ	611μ	452μ	$16.0m$	$957m$	464μ	543μ	617μ	442μ	$14.3m$	$849m$
PM ($^\circ$)	63.2	70.3	82.6	58.4	68.5	X	62.1	69.7	82.0	57.6	67.7	80.3
Tset (ns)	0.87	1.5	3.4	0.37	1.13	3.16	0.85	1.46	3.3	1.18	6.74	X
Swing (V)	0.66	0.84	0.95	0.64	0.77	0.86	0.78	0.93	X	X	0.82	0.98

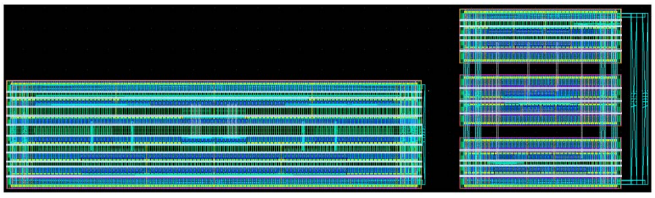

Fig. 20. BAG-generated LVS-passed layout for the core and biasing folded-cascode circuit for an obtained performance.

TABLE XIV
RSC FOR AUTOCKT WITH VARYING SEEDS (AVERAGE MIN/MEAN/MAX) AND TWO PRIOR WORKS

Experiment	Schematic RSC	RSC PEX	Obt. Layout Target?
AutoCkt, Seed 1	4 / 30 / 40	4	Yes
AutoCkt, Seed 2	3 / 27 / 40	4	Yes
AutoCkt, Seed 3	3 / 25 / 40	4	Yes
[16]	-	23	Yes
[10] (schematic)	10012	-	No
[30] (schematic)	58844	-	No

TABLE XV
8-BIT CORNER SIMULATION RESULTS

Metric	Target	FF	FS	SF	SS
UGBW (GHz)	> 0.98	3.15	2.93	3.13	2.95
Gain (dB)	> 54	57.2	57.3	60.1	56.5
Noise (μV_{rms})	$< 750 \pm 40\%$	579	589	585	591
Power (μA)	minimize	538	471	522	459
PM ($^\circ$)	> 60	74.0	70.0	73.2	68.9
Tset (ns)	< 1.7	1.44	1.56	1.41	1.61
Swing (V)	> 0.8	0.86	0.79	0.93	0.79

TABLE XVI
10-BIT CORNER SIMULATION RESULTS

Metric	Target	FF	FS	SF	SS
UGBW (GHz)	> 1.2	4.27	4.10	4.25	4.01
Gain (dB)	> 60	59.1	60.0	61.6	64.7
Noise (μV_{rms})	$< 380 \pm 40\%$	637	567	600	551
Power (μA)	Minimize	514	518	520	525
PM ($^\circ$)	> 60	73	68.7	72.5	68.6
Tset (ns)	< 1.7	1.38	0.67	1.36	1.65
Swing (V)	> 0.8	0.79	0.77	0.94	0.91

understands. If we had simply taken the first schematic simulation parameters that met the target and used this to run PEX, depicted as the theoretical schematic simulation in Fig. 19, the target specification would not have been reached.

3) AutoCkt With Layout Parasitics: We test the combined schematic and layout deployment methodology by giving the agent a range of target design specifications to reach. Table XI shows the minimum, maximum, and mean obtained layout performances on 60 trajectories. We find that our framework successfully reaches a range of valid PEX performances, meaning that it successfully sizes parameters even when the change in distribution from layout parasitics is significant.

4) Pipeline ADC Target Specifications: We select the two target specifications for the 8 and 10 bit pipeline ADC (Table V) and input them to the trained agent during deployment. The results show that our framework obtains LVS-passed designs that meet all target metrics for the 8-bit ADC, and all metrics for the 10-bit ADC when noise variability is taken into account. The obtained performances are summarized in Table XII, and the BAG-generated folded-cascode amplifier and biasing layout is shown in Fig. 20. Table XII also shows the layout performances obtained by two prior works run with schematic simulations, whose converged designs are then taken and simulated with layout in BAG. Despite both works obtaining designs that meet the target in schematic, they fail to meet or obtain valid performances once parasitics are considered. We note that the most efficient schematic RSC solution from [7] would take almost 13.9 days to converge using layout parasitic simulations with the same RSC.

Our RL algorithm handles the distribution difference between schematic and layout by requiring additional simulations in schematic to meet the target, on average taking 30 schematic simulation steps and four parasitic steps across the 60 tested target specifications, shown in Table XIV.

TABLE XVII
COMPARISON TABLE FOR PRIOR ANALOG AUTOMATION WORKS

	[21]	[23]	[22]	[14]	[15]	[10]	**This Work**
Algorithm	Metamodel	Metamodel	Metamodel	GCN+RL	RL+starting pt.	NN+GA	RL
Technology	$45nm$	$90nm$	$90nm$	$180nm$	$6nm$	$14nm$	$16nm$ FF
Circuit Complexity (# params, #metrics)	$6, 2$	$5, 1$	$6, 2$	$25, 5$	$4, 2$	$17, 9$	$29, 7$
Training cost	Low	Low	Low	High	Medium	Medium	High
Considers parasitics	Yes	Yes	Yes	No	No	Yes	Yes
Uses parasitics model	Yes	Yes	Yes	No	No	No	No
Process corners	No	Yes	Yes	No	Yes	No	Yes*
RSC	2	2	2	10000	2609	435	4

Validated on process corners, but not trained

Our approach reliably reaches target specifications in layout, with state-of-the-art PEX simulation RSC, beating [16] by almost 6*X* in a more complex circuit topology. We note that prior work would have required too many RSCs to deploy with layout simulations, as their RSCs in schematic are too high.

5) PEX Corner Simulation Results: To determine if AutoCkt reaches design points that are robust enough once process corners are considered, corner analyses is run for each of the 60 performances the agent obtained. To understand the validity of these results, which are shown in Table XIII, we calculate two metrics: 1) how many of the 60 design points still have valid results with corner simulations, meaning the metrics are within the same region as the nominal corner and 2) are the designs that satisfy the 8 and 10 bit pipeline ADC requirement valid once corners are considered.

We find that 96.7% of the obtained performances are valid, with just two of the 60 performances resulting in invalid corner simulation results. This shows that despite not considering corner variation in its sizing, the agent does very well in reaching design points that are stable.

We also show the corner simulation results for our previous 8 and 10 bit ADC designs in Tables XV and XVI. For both designs, the target specifications are satisfied for every metric, with the exception of noise in the 10-bit ADC design. We again posit that our noise metric is an estimated value. We run Monte Carlo simulations on these designs as well, but find that the results varied considerably.

To incorporate process corner/Monte Carlo PVT simulations, we propose using accurate estimation tools, such as [22], [31], and [32] to predict these variations during training with schematic simulations, giving the agent the mean and standard deviation of each metric performance as part of the state, or use a similar deployment generalization methodology proposed in Section II to run process corner/Monte Carlo simulations only when the agent has obtained a valid schematic and layout performance design. If the results of these simulations do not meet a predefined acceptable variation, the worst performance is given back to the agent to once again go through the schematic and layout sizing procedure.

6) Comparison to Prior Analog Sizing Work: We compare AutoCkt to prior analog sizing works in Table XVII. In this comparison, we take the best performing, most complex circuit topologies tested by each prior work, and include their circuit complexity measured by the number of parameters in the circuit and number of metrics tested, whether parasitics are considered, whether a parasitics model is used, process corners are considered, and the RSC. We show that our work tests on a complex circuit while considering true layout parasitic simulations, and has one of the lowest deployment RSC values.

V. CONCLUSION AND FUTURE WORK

In this article we expand our results on the RL tool that sizes analog circuits, AutoCkt [16], to show that this framework is scalable, accurate, and reliable. We test our framework on a two-stage transimpedance amplifier and two-stage folded cascode with biasing, and demonstrate that our tool achieves state-of-the-art runtime RSC, up to 38× more efficient than prior work. We show that our algorithm is robust, generalizes to an environment that considers layout parasitics, and can handle the large amount of error present between post-layout and schematic-only simulations. With our combined schematic and layout parasitic simulation approach, achieved by using the Berkeley Analog Generator, we find that our approach requires, on average, four layout simulations to converge to a performance that meets a target specification. In addition, our obtained LVS-passed folded-cascode performances meet target 8 and 10-bit pipeline ADC designs that are valid when process corners are considered. We also conduct extensive analysis on where our framework fails, and why this machine learning approach can be practically incorporated to an existing analog sizing flow.

REFERENCES

[1] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "IPRAIL—Intellectual property reuse-based analog IC layout automation," *Integration*, vol. 36, no. 4, pp. 237–262, Nov. 2003.

[2] P. G. A. Jespers and B. Murmann, *Systematic Design of Analog CMOS Circuits*. Cambridge U.K.: Cambridge Univ. Press, 2017.

[3] E. Berkcan and M. D'Abreu, "Physical assembly for analog compilation of high voltage ICs," in *Proc. IEEE Cust. Integr. Circuits Conf.*, Rochester, NY, USA, 1988, pp. 1–7.

[4] K. Mendhurwar, H. Sundani, P. Aggarwal, R. Raut, and V. Devabhaktuni, "A new approach to sizing analog CMOS building blocks using pre-compiled neural network models," *Analog Integr. Circuits Signal Process.*, vol. 70, no. 3, pp. 265–281, Mar. 2012.

[5] W. Daems, G. Gielen, and W. Sansen, "An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits," in *Proc. Design Autom. Conf.*, New Orleans, LA, USA, 2002, pp. 431–436.

[6] M. W. Cohen, M. Aga, and T. Weinberg, "Genetic algorithm software system for analog circuit design," *Procedia CIRP*, vol. 36, pp. 17–22, Jan. 2015.

[7] 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.

[8] P. Terway, K. Hamidouche, and N. K. Jha, "DISPATCH: Design space exploration s Sep. 2020. [Online]. Available: arXiv2009.10214.
[9] G. Wolfe and R. Vemuri, "Extraction and use of neural network models in automated synthesis of operational amplifiers," *IEEE Trans. Comput. Design Integr. Circuits Syst.*, vol. 22, no. 2, pp. 198–212, Feb. 2003.
[10] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović, "BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks," in *Proc. IEEE/ACM Int. Conf. Comput. Design*, Westminster, CO, USA, Nov. 2019, pp. 1–8.
[11] Y. Li, Y. Wang, Y. Li, R. Zhou, and Z. Lin, "An artificial neural network assisted optimization system for analog design space exploration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2640–2653, Oct. 2020.
[12] W. Lee and F. A. Oliehoek, "Analog circuit design with Dyna-style reinforcement learning," Nov. 2020. [Online]. Available: arXiv2011.07665.
[13] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," Dec. 2018. [Online]. Available: arXiv1812.02734.
[14] H. Wang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. 57th ACM/IEEE Design Autom. Conf.*, Jul. 2020, pp. 1–6.
[15] K.-E. Yang *et al.*, "Fast design space adaptation with deep reinforcement learning for analog circuit sizing," 2020. [Online]. Available: arXiv2009.13772.
[16] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2020, pp. 490–495.
[17] D. J. Chen and B. J. Sheu, "Automatic layout synthesis of analog ICs using circuit recognition and constraint analysis techniques," *Analog Integr. Circuits Signal Process.*, vol. 1, no. 1, pp. 75–87, Mar. 1991.
[18] N. Lourenço, R. Martins, and N. Horta, "Layout-aware sizing of analog ICs using floorplan & routing estimates for parasitic extraction," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Grenoble, France, 2015, pp. 1156–1161.
[19] H. Habal and H. Graeb, "Constraint-based layout-driven sizing of analog circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 8, pp. 1089–1102, Aug. 2011.
[20] I.-L. Tseng, A. Postula, and L. Jozwiak, "Symbolic extraction for estimating analog layout parasitics in layout-aware synthesis," in *Proc. IEEE Mixdes*, 2005, pp. 195–199.
[21] O. Garitselov, S. P. Mohanty, and E. Kougianos, "A comparative study of metamodels for fast and accurate simulation of nano-CMOS circuits," *IEEE Trans. Semicond. Manuf.*, vol. 25, no. 1, pp. 26–36, Feb. 2012.
[22] S. P. Mohanty and E. Kougianos, "Incorporating manufacturing process variation awareness in fast design optimization of nanoscale CMOS VCOs," *IEEE Trans. Semicond. Manuf.*, vol. 27, no. 1, pp. 22–31, Feb. 2014.
[23] D. Ghai, S. P. Mohanty, and E. Kougianos, "Design of parasitic and process-variation aware Nano-CMOS RF circuits: A VCO case study," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 9, pp. 1339–1342, Sep. 2009.
[24] R. Castro-Lopez, O. Guerra, E. Roca, and F. V. Fernandez, "An integrated layout-synthesis approach for analog ICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1179–1189, Jul. 2008.
[25] O. Garitselov, S. P. Mohanty, and E. Kougianos, "Fast-accurate non-polynomial metamodeling for nano-CMOS PLL design optimization," in *Proc. IEEE Int. Conf. VLSI Design*, 2012, pp. 316–321.
[26] O. Garitselov, S. P. Mohanty, E. Kougianos, and O. Okobiah, "Metamodel-assisted ultra-fast memetic optimization of a PLL for WiMax and MMDS applications," in *Proc. Int. Symp. Qual. Electron. Design (ISQED)*, 2012, pp. 580–585.
[27] E. Liang *et al.*, "RLlib: Abstractions for distributed reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, Dec. 2017, pp. 3059–3068.
[28] E. Chang *et al.*, "BAG2: A process-portable framework for generator-based AMS circuit design," in *Proc. IEEE Cust. Integr. Circuits Conf.*, San Diego, CA, USA, Apr. 2018, pp. 1–8.
[29] N. Bako, Ž. Butković, and A. Barić, "Design of fully differential folded cascode operational amplifier by the gm/ID methodology," in *Proc. 33rd Int. Conv. Inf. Commun. Technol. Electron. Microelectron.*, 2010, pp. 89–94.
[30] D. Whitley, "A genetic algorithm tutorial," *Stat. Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994.
[31] I. C. Odabasi, M. B. Yelten, E. Afacan, F. Baskaya, A. E. Pusane, and G. Dundar, "A rare event based yield estimation methodology for analog circuits," in *Proc. IEEE 21st Int. Symp. Design Diagnos. Electron. Circuits Syst.*, Budapest, Hungary, Apr. 2018, pp. 33–38.
[32] M. Ahuja, S. Narang, and S. Patnaik, "A process corner detection methodology for resilience towards process variations using adaptive body bias," in *Proc. IEEE Int. Conf. Circuit Power Comput. Technol. (ICCPCT)*, Nagercoil, India, 2015, pp. 1–6.

Keertana Settaluri (Member, IEEE) received the B.S. degree in electrical engineering from the University of Florida, Gainesville, FL, USA, in 2015. She is currently pursuing the Ph.D. degree in electrical engineering with the University of California at Berkeley, Berkeley, CA, USA, on applications of machine learning to accelerate ASIC design.

Ms. Settaluri was a recipient of the Qualcomm Innovation, SanDisk, and Oldham Fellowships.

Zhaokai Liu (Member, IEEE) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2017, and the M.S. degree in electrical engineering from the University of California at Berkeley, Berkeley, CA, USA, in 2020, where he is currently pursuing the Ph.D. degree.

His current research interests include mixed-signal circuits, high-speed A/D converters, and analog circuit design automation.

Rishubh Khurana (Member, IEEE) received the B.E. degree in electrical, electronics, and communications engineering from the Netaji Subhas Institute of Technology, New Delhi, India, in 2007.

He is working as a Machine Learning Researcher with Qualcomm, Inc., Bengaluru, India.

Arash Mirhaj, photograph and biography not available at the time of publication.

Rajeev Jain (Fellow, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Delhi, New Delhi, India, in 1978, and the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven, Leuven, Belgium, in 1985.

He is an Emeritus Professor of Electrical Engineering with the University of California at Los Angeles, Los Angeles, CA, USA, and a Senior Director with Qualcomm Research, San Diego, CA, USA.

Dr. Jain awards include the Allied Signal Faculty Research Award and the Young Faculty Award from Northrop Grumman Industries.

Borivoje Nikolic (Fellow, IEEE) received the Dipl.Ing. and M.Sc. degrees in electrical engineering from the University of Belgrade, Belgrade, Serbia, in 1992 and 1994, respectively, and the Ph.D. degree from the University of California at Davis, Davis, CA, USA, in 1999.

He joined the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA, USA, in 1999, where he is currently a Professor. His current research interests include digital, analog, and RF integrated circuit design, and VLSI implementation of communications and signal processing algorithms.