CRONuS: Circuit Rapid Optimization with Neural Simulator

Youngmin Oh, Doyun Kim, Yoon Hyeok Lee, and Bosun Hwang
Samsung Advanced Institute of Technology, Republic of Korea
{youngmin0.oh, doyun.d.kim, yh.luke.lee, bosun.hwang}@samsung.com

***Abstract*—Automation of analog circuit design is highly desirable, yet challenging. Various approaches such as deep reinforcement learning (DRL), genetic algorithms, and Bayesian optimization have been proposed and found to be effective. However, these techniques require a large number of interactions with a real simulator, leading to high computational costs. Therefore, we present a novel DRL method, CRONuS, for automatic analog circuit design that uses a surrogate for the simulator. With the help of the surrogate, our method is capable of augmenting a data set with a conservative reward design for stable policy training, without having to interact with the simulator. Regardless of the type of analog circuit, our experiment demonstrated a more than $5\times$ improvement in sample efficiency with varying target performance metrics.**

***Index Terms*—Deep Reinforcement Learning, DRL, Model-based DRL, MbDRL, Transistor sizing, Sample Efficiency, Robust Circuit Design**

I. Introduction

Designing analog circuits is still one of the challenging tasks in the chip design process. It requires engineers to choose the parameters of multiple transistors and circuit components with a given circuit topology to meet design objectives such as gain and bandwidth. The analog design tasks involve numerous design iterations, causing long design time, due to the vast design space of numerous parameters and the intricate trade-offs between performance metrics.

To address the problem, various design automation methods such as the genetic algorithm and Bayesian optimization [1–3] have been suggested. In particular, deep reinforcement learning (DRL) has been widely adapted in the area of analog circuit design, as it can automatically learn to adjust design parameters to meet the desired specifications in analog and mixed-signal circuit designs [4–12]. This is due to its capability to learn even extremely complex tasks with generalizability, e.g., Go [13]. However, the success of current DRL techniques in analog circuit design depends on high simulation costs, i.e., low sample efficiency. For example, [6] reported that more than 160,000 interactions with a simulator were required to optimize a two-stage operational amplifier [14] to meet varying target specifications.

We present *Circuit Rapid Optimization with Neural Simulator* (CRONuS), a DRL method to overcome the limitation, i.e., low sample efficiency, of previous automation approaches. As in Fig. 1, CRONuS uses an ensemble of neural networks as a neural simulator to replace the real simulator by learning the behavior of it in a supervised learning manner with a collection of real circuit parameters and performance metrics. The neural

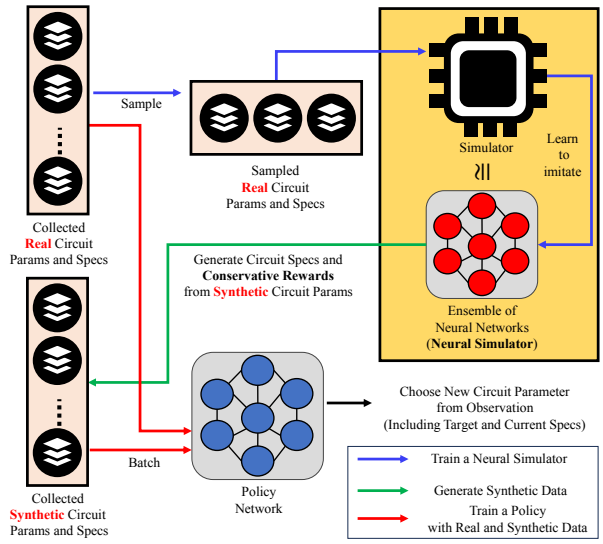


Fig. 1. CRONuS's Overview: 1) The neural simulator learns in a supervised manner from data collected by a real simulator (**blue lines**). 2) The neural simulator generates synthetic data indistinguisable from the real data (**green lines**) with new conserative rewards. 3) A policy network is trained with real and synthetic data (**red lines**).

simulator provides the distribution of each circuit's performance metric prediction and reward prediction by mean and standard deviation, so we can design a conservative reward for stable policy training. To be specific, the standard deviation indicates how certain the performance prediction is, so a conservative reward is defined by the values that deviate from the mean of the prediction by a confidence coefficient multiplied by the standard deviation.

By sampling with conservative rewards through performance distributions, CRONuS is able to create a large number of synthetic data for consistent and rapid policy training. As a result, CRONuS significantly reduces interactions with the real simulator to train a policy network, implying the high sample efficiency of CRONuS. Furthermore, thanks to its ability to learn distributions, the neural simulator can also predict local process variation, altering the parameters of each transistor in a random manner, which is beneficial for training a robust policy network concerning local variation. To validate the effectiveness of CRONuS, particularly in sample efficiency, we compare it with other DRL approaches, on different circuits, for static and transient analysis, under varying and fixed target specifications.

Contribution. To design analog circuits, CRONuS uses

979-8-3503-4859-0/DATE24/© 2024 EDAA

an ensemble of neural networks as a surrogate for a real simulator to mimic its behavior. Therefore, CRONuS generates a large amount of synthetic data with conservative rewards that encourage stable policy training. To the best of our knowledge, this is the first model-based DRL framework that demonstrates remarkable sample efficiency regardless of the type of circuit. In fact, we observed over $5\times$ improvement in sample efficiency compared to existing methods with variable target specifications. We also experimentally confirmed that CRONuS is useful for learning a robust policy on local process variation.

II. Related Results

1) Deep Reinforcement Learning (DRL): DRL is divided into model-free and model-based algorithms. Model-free DRL algorithms, such as DDPG [15], PPO [16], TD3 [17], and SAC [18], do not require a model of the environment; they only learn the optimal policy through direct interaction with the environment, so they require a large number of interactions with the environment. On the other hand, model-based DRL algorithms [13, 19, 20] employ a model of the environment to mimic the behavior of the environment, and the acquired model is used as a surrogate for the environment. The surrogate is capable of generating synthetic data that are indistinguishable from the real data if it is properly trained. This means that the policy using the synthetic data requires only a few direct interactions with the environment, thus reducing the number of direct interactions with the environment. CRONuS adapts the surrogate concept as well.

2) Circuit Design Automation: Genetic algorithm (GA) is a popular choice to find the best candidates to maximize a given fitness function by repeating the mutation and crossover [1, 2]. However, GA requires a large number of interactions with the real environment, making it less efficient for complex tasks. An alternative to GA is Bayesian optimization (BO) [3]. BO uses the Bayes rule to mathematically obtain posterior knowledge from prior knowledge and predict candidate values that maximize the objective function. Unfortunately, BO is not suitable for high-dimensional domains because its computational cost scales as the number of parameters. To address the limitations of each method, recent approaches have incorporated deep neural networks and achieved impressive results. In particular, DRL has been widely used to obtain generalizable analog circuit designs. DRL has demonstrated its ability to transfer knowledge between different technologies [4, 5] and even between different topologies [5]. DRL algorithms also generally handle different target specifications [6]. Recently, DRL has been used to design analog circuits under process variations that randomly change transistor values [8, 10]. [21] used a neural network to predict a reward as part of a model-based DRL approach. However, the DRL formulation in their work was not well defined, as the input to the policy network was simply a random value sampled from a standard normal distribution, rather than a clearly defined observation. Furthermore, they did not compare the performance of their approach with other algorithms. Finally, it is unable to effectively control the circuit parameters under changing target specifications.

III. Main Method

A. DRL Design

It is essential to formulate appropriate observation, action and reward, as these will allow the policy network to be trained in a stable way and to make reasonable decisions.

1) Observation: Let N be the number of circuit parameters, such as transistor width, and K be the number of performance metrics, such as gain. For a fixed size of vector $\mathbf{m^g} = \{m_1^g, \cdots, m_K^g\}$, we define an observation $\mathbf{o}$ as a vector,

$$\mathbf{o} = (\oplus_{i=1}^{N} p_i) \oplus (\oplus_{i=1}^{K} \mathrm{d}(m_i, m_i^g)) \oplus (\oplus_{i=1}^{K} \mathrm{d}(m_i^*, m_i^g)), \quad (1)$$

with the following relative difference function: $\mathrm{d}(x, y) = \frac{x-y}{x+y}$, where the current design parameters $\mathbf{p} = \{p_1, \cdots, p_N\}$, the current performance metrics $\mathbf{m} = \{m_1, \cdots, m_K\}$, the target specifications $\mathbf{m}^* = \{m_1^*, \cdots, m_K^*\}$, and the concatenation operator $\oplus$. This definition is the same as in [6]

2) Action: CRONuS's action represents how much the circuit parameters change ($\Delta\mathbf{p}$), that is, $\mathbf{p}_{new} = \mathbf{p}_{old} + \Delta\mathbf{p}$ since fine adjustment of the parameters helps to find the optimal value efficiently. We appropriately scaled the action for each update of the parameters of CRONuS to a range of $[-1, 1]^N$, so that a learnable agent can cover a maximum and minimum range of circuit parameters up to 30 simulation steps when scaling back to the original values.

3) Reward: We employ the reward function r (Figure of Merit) as [6]:

$$r\left(\left(\oplus_{i=1}^{K} m_i\right) \oplus \left(\oplus_{i=1}^{K} m_i^*\right)\right) = \begin{cases} \mathtt{FoM} & \text{if } \mathtt{FoM} < -0.01, \\ 10 & \text{if } \mathtt{FoM} \geq -0.01, \end{cases} \quad (2)$$

$$\mathtt{FoM} = \sum_{i=1}^{K} \min\left\{\mathrm{d}(m_i, m_i^*), 0\right\}.$$

Therefore, the maximum reward is only gained if all the target specifications are met.

B. Neural Simulator

1) Architecture: We create a neural simulator using an ensemble of fully connected neural networks $\{N_{\theta_i}\}_{i=1}^{E}$ with learnable parameters θ_i. Each network N_{θ_i}, taking the current observation o and the action a as input defined in Eq. (1), learns the Gaussian distribution of the reward and the difference $o' - o$, instead of the next observation o' for practical training. Indeed, since we already know o, learning the difference is equivalent to training to predict performance metrics. In other words, N_{θ_i} predicts the means $\mu_i = \{\mu_{i,j}\}_{j=0}^{|o|}$, and the standard deviations $\sigma_i = \{\sigma_{i,j}\}_{j=0}^{|o|}$ for the reward ($j = 0$) and the difference ($1 \leq j \leq |o|$), where $\mu_{i,j} \in \mathbb{R}$ and $\sigma_{i,j} \in \mathbb{R}$ for $0 \leq j \leq |o|$.

2) Train: At the neural simulator training phase, CRONuS collects a data set by interactions with the real simulator in the real buffer $B_r = \{o, a, o', r, d\}$, and divides it randomly into a training set $B_{r,t}$ and a validation set $B_{r,v}$. Subsequently, each N_{θ_i} is updated with the following loss function.

$$L_{\{N_{\theta_i}, B_{r,t}\}_{i=1}^{E}} = \sum_{i=1}^{E} L_{N_{\theta_i}, B_{r,t}}, \quad (3)$$

Algorithm 1 CRONuS's training process

Initialize buffers: $B_r = \{\}$, $B_s = \{\}$;
Initialize a neural simulator $\{N_{\theta_i}\}_{i=1}^{E}$, policy network;
Set $\mathbf{m}^g = \{m_1^g, \cdots, m_K^g\}$;
for timestep **do**
  Reset target specifications **m***;
  if timestep $\equiv 0 \pmod{\text{train\_freq}}$ **then**
    Train $\{N_{\theta_i}\}_{i=1}^{E}$ via B_r with the loss by Eq. (3);
    Set $B_s = \{\}$
    Determine initial observations sampled in B_r;
    Generate **synthetic** data set with conservative rewards by Eq. (4)-(5) with the $\{N_{\theta_i}\}_{i=1}^{E}$ and policy network;
    Push generated data set to B_s;
  if timestep $>$ threshold **then**
    Do action a by the policy network;
  else
    Do action a at random;
    Train the agent by samples in $B_s \bigcup B_r$;
  Run a **real** simulator to get **real** data and push it to B_r;

where

$$L_{N_{\theta_i},B_{r,t}} = \mathbb{E}_{\{o,a,r,o'\}\sim B_{r,t}}[(|\mu_i - r_j \oplus (o' - o)|^2 + \sigma_i \log \sigma_i^2)/\sigma_i],$$
$$N_{\theta_i}(o,a) = (\mu_i, \sigma_i) \in \mathbb{R}^{|o|+1} \times \mathbb{R}^{|o|+1}.$$

The training phase is completed when the loss $L_{\{N_{\theta_i},B_{r,v}\}}$ stops decreasing for a predetermined number of steps. The elite networks among the ensemble are then chosen since they are used later to generate synthetic data by rolling-out.

3) Roll-out: After training the neural simulator, a synthetic buffer B_s is initialized that collects a synthetic data set. Specifically, CRONuS samples observations from B_r, which are then used as initial observations. Then the policy network and the neural simulator interact with each other through rolling out. The policy network uses each observation to predict the appropriate action a. The next observation prediction $\widehat{o}'$ and the conservative reward $\widehat{r}$ are then determined by means $\{\mu_i\}_{i\in\text{elites}}$ and standard deviations $\{\sigma_i\}_{i\in\text{elites}}$ (chosen in Section III-B2), as follows.

$$\widehat{o}' = o + \mathbb{E}_{i\in\text{elites},1\le j\le|o|}[\mu_{i,j} + \epsilon\sigma_{i,j}], \quad (4)$$
$$\widehat{r} = \mathbb{E}_{i\in\text{elites}}[\mu_{i,0} - \zeta\sigma_{i,0}] \quad (5)$$

where $\epsilon \sim \mathcal{N}(0,1)$, and $0l\zeta$ is the confident coefficient. $\widetilde{r}$ by Eq. (5) can be regarded as the conservative reward from the predicted reward distribution. The policy then regards the next observation prediction $\widehat{o}'$ obtained by Eq. (4) as the current observartion o to select the actions again. This cycle is repeated for a predefined roll-out length, and the collected data are pushed into B_s.

C. Policy Model

We employ double critic networks to reduce the overestimation of Q-values [17] and a long-short-term memory (LSTM) [22] as the policy network architecture to select each component variation in sequence. For each simulation step, the policy is repeatedly trained based on SAC [18] for the predefined gradient steps, using the data sampled from $B_r \bigcup B_s$, where the real data ratio is predefined (see Table I).

IV. Experiments

A. Benchmark Tasks & Algorithms

We validate CRONuS on the following benchmarks: TwoStageOpAmp, RevTwoStageOpAmp where NMOS and PMOS exchanges in TwoStageOpAmp, TwoStageTIA, ThreeStageTIA, Comparator, and LevelConverter. Fig. 2 shows the detailed topology of the benchmarks. When considering varying target specifications, the randomly chosen target specifications in Table II are given for each episode consisting of 30 steps, and the policy must find the circuit parameters such as the widths of transistors and capacitors to meet all targets.

We compare the CRONuS with the popular DRL algorithms PPO [16], TD3 [17], and SAC [18] under default hyperparameters therein. Here, PPO has been widely employed in analog circuit design, as seen in [6, 23]. TD3 is an extension of DDPG [15], which has been frequently utilized for analog circuit designs, as demonstrated in [4, 5, 10]. SAC is a key policy learning algorithm in CRONuS. Furthermore, we consider SAC-BO, a combination of SAC and BO, where BO is used to configure the circuit parameters, and then SAC further optimizes them as [10] in scenarios where the target specifications remain constant. Table I introduces the main hyperparameters of our experiments. The other hyperparameters are chosen as the default parameters in [19].

TABLE I
Main Hyperparameters for Neural Simulator

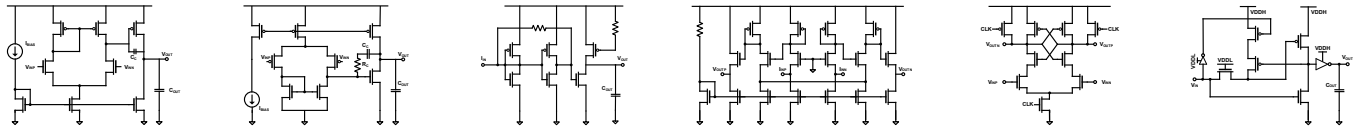
Parameter	Value
Real Data Ratio	0.05
# of Ensemble Networks (Elites)	7 (5)
Network Architecture	Fully-Connected
Network Size	200-200-200-200-200
Model Training Frequency	15 simulator runs
Roll-out Length	15 steps
Gradient Steps	20 per simulator run
Training Stop Criterion	50 validations
Roll-out Batchsize	# of Collected Real Data

B. Sample Efficiency Evaluations

1) Performance of Neural Simulator: It is critical for a neural simulator to accurately reproduce the behavior of the simulator. To this end, 1,200 simulation results were used to train the neural simulator. Fig. 3 shows the accuracy of the neural simulator's prediction about performance metrics of the TwoStageOpAmp for 300 circuit parameter data, which was not used for training. It is clear that the neural simulator correctly predicts each trend. The results in Table III demonstrate that the trend is consistent regardless of the circuits. Thus, the neural simulator can be trusted to allow the policy network to learn efficiently under surrogate data sets.

TABLE II
CIRCUIT TARGET SPECIFICATIONS AND RANGE OF TRANSISTOR WIDTH AND CAPACITOR VALUE

Circ. for Static Analysis	Gain (V/V)	UGBW (Hz)	PM (°)	BC (mA)	Width (Capacitor) Range (um (pF))
TwoStageOpAmp (45nm)	[2.0e2, 4.0e2]	[1.0e6, 2.5e7]	60	[1.0e-4, 1.0e-2]	[1.0e-1, 3.0e1]
RevTwoStageOpAmp (45nm)	[2.0e2, 4.0e2]	[1.0e6, 2.5e7]	60	[1.0e-4, 1.0e-2]	[1.0e-1, 3.0e1]
TwoStageTIA (45nm)	[1.0e2, 1.0e3]	[4.0e9, 5.0e9]	60	[1.0e-2, 2.0e-2]	[1.0e-1, 1.0e2]
ThreeStageTIA (45nm)	[2.0e2, 3.0e2]	[1.0e8, 1.0e9]	60	[1.0e-3. 1.0e-2]	[1.0e-2, 3.0e1]
Circ. for Transient Analysis	**Avg. Power (W)**	**Ratio**	**Avg. Delay (sec)**	**Delay Balance (sec)**	**Width (Capacitor) Range (um (pF))**
Comparator (45nm)	[6.5e-11, 7.0e-11]		[5.0e-9, 5.5e-9]		[1.0e-1, 3.0e1]
LevelConverter (45nm)	[0.0, 2.0e-6]	[0.999, 1.0]	[4.0e-8, 6.0e-8]	[1.0e-12, 1.0e-10]	[4.6e-1, 1.0e2]



(a) (b) (c) (d) (e) (f)

Fig. 2. Benchmark Circuits: (a) TwoStageOpAmp, (b) RevTwoStageOpAmp, (c) TwoStageTIA, (d) ThreeStageTIA, (e) Comparator, (f) LevelConverter

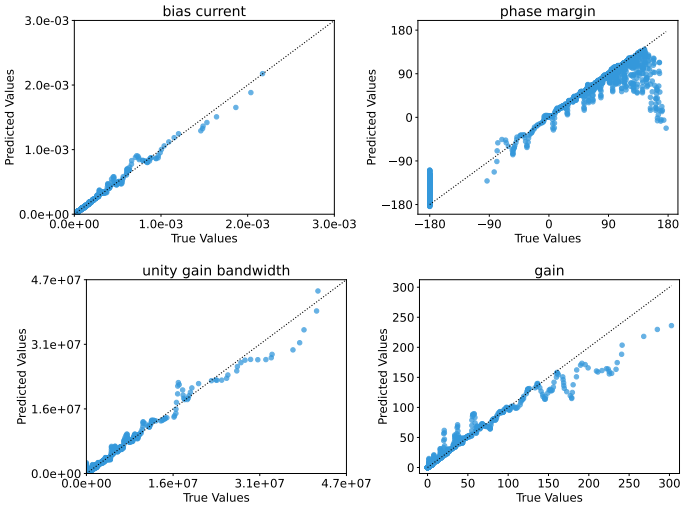


Fig. 3. Neural Simulator's Prediction of TwoStageOpAmp Performance.

TABLE III
METRICS OF PREDICTING EACH CIRCUIT'S SPECIFICIATIONS

Circuits	MSE	RMSE	MAE	R2
TwoStageOpAmp (45nm)	0.0054	0.0736	0.0118	0.7917
RevTwoStageOpAmp (45nm)	0.0034	0.0583	0.0091	0.8134
TwoStageTIA (45nm)	0.0064	0.0803	0.0167	0.7743
ThreeStageTIA (45nm)	0.0057	0.0755	0.0223	0.8477
Comparator (45nm)	0.0011	0.0338	0.0172	0.8408
LevelConverter (45nm)	0.0101	0.1005	0.0206	0.7316

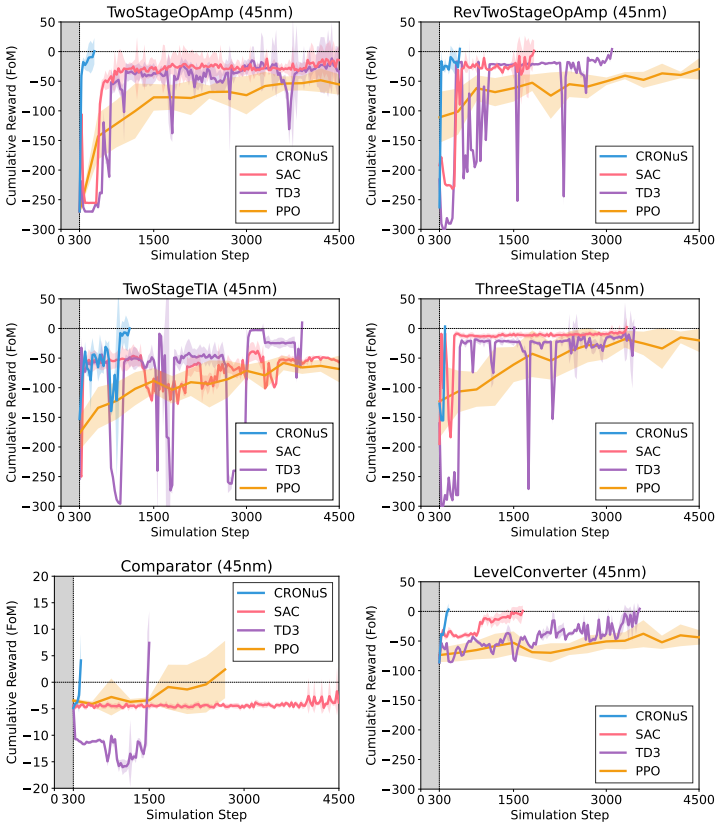


Fig. 4. Learning curves under varying target specifications. Each algorithm is terminated early if the average cumulative rewards for an episode exceed 0. The gray zone (300 steps) denotes the exploration phase. The solid lines and the shaded areas in all experimental results show the mean and standard deviations of five trials with randomly generated seeds.

2) Performance of Policy Networks: Table IV and Table V show performance comparisons among DRL algorithms for varying target specifications (up to 4,500 simulation steps) and fixed ones (up to 3,000 simulation steps), respectively. If the average cumulative rewards for varying target specifications (or the maximum rewards for fixed target specifications) are greater than zero, the algorithms stop training as [6]. To validate each method, we calculated the relative sample efficiency based on our method CRONuS, whose value was set to 1.0. If any of the algorithms could not succeed within 4,500 (3,000) simulation steps, the relative sample efficiency values were evaluated based on 4,500 (3,000) simulation steps in Table IV (Table V). Furthermore, Fig. 4 (Fig. 5) shows the learning curves of each algorithm for different tasks. It is shown that the sample efficiency of CRONuS outperforms other popular

TABLE IV
RELATIVE SAMPLE EFFICIENCY OF ALGORITHMS UNDER VARYING TARGET SPECIFICATIONS TILL 4,500 SIMULATION STEPS

Algorithm	CRONuS	SAC	TD3	PPO
TwoStageOpAmp (45nm)	1.0 (540 steps)	≤0.12 (4,500 steps)	≤0.12 (4,500 steps)	≤0.12 (4,500 steps)
RevTwoStageOpAmp (45nm)	1.0 (630 steps)	0.34 (1,830 steps)	0.20 (3,090 steps)	≤0.14 (4,500 steps)
TwoStageTIA (45nm)	1.0 (1,110 steps)	≤0.25 (4,500 steps)	0.28 (3,900 steps)	≤0.24 (4,500 steps)
ThreeStageTIA (45nm)	1.0 (390 steps)	0.12 (3,330 steps)	0.11 (3,450 steps)	≤0.09 (4,500 steps)
Comparator (45nm)	1.0 (420 steps)	≤0.09 (4,500 steps)	0.28 (1,500 steps)	0.16 (2,700 steps)
LevelConverter (45nm)	1.0 (450 steps)	0.27 (1,650 steps)	0.13 (3,540 steps)	≤0.10 (4,500 steps)
Average	**1.0**	≤0.20	≤0.19	≤0.15

TABLE V
RELATIVE SAMPLE EFFICIENCY OF ALGORITHMS UNDER FIXED TARGET SPECIFICATIONS TILL 3,000 SIMULATION STEPS

Algorithm	CRONuS	SAC w/ BO	SAC	TD3	PPO
TwoStageOpAmp (45nm)	1.0 (450 steps)	≤0.15 (3,000) steps	0.37 (1,230 steps)	≤0.15 (3,000) steps	≤0.15 (3,000) steps
RevTwoStageOpAmp (45nm)	1.0 (330 steps)	≤0.11 (3,000 steps)	0.46 (720 steps)	0.42 (780 steps)	≤0.11 (3,000 steps)
TwoStageTIA (45nm)	1.0 (990 steps)	≤0.33 (3,000 steps)	≤0.33 (3,000 steps)	≤0.33 (3,000 steps)	≤0.33 (3,000 steps)
ThreeStageTIA (45nm)	1.0 (390 steps)	0.29 (1,350 steps)	≤0.13 (3,000 steps)	0.16 (2,490 steps)	≤0.13 (3,000 steps)
Comparator (45nm)	1.0 (390 steps)	0.68 (570 steps)	≤0.13 (3,000 steps)	0.33 (1,200 steps)	0.43 (900 steps)
LevelConverter (45nm)	1.0 (420 steps)	0.47 (900 steps)	0.39 (1,080 steps)	0.30 (1,410 steps)	≤0.14 (3000 steps)
Average	**1.0**	≤0.34	≤0.31	≤0.29	≤0.22

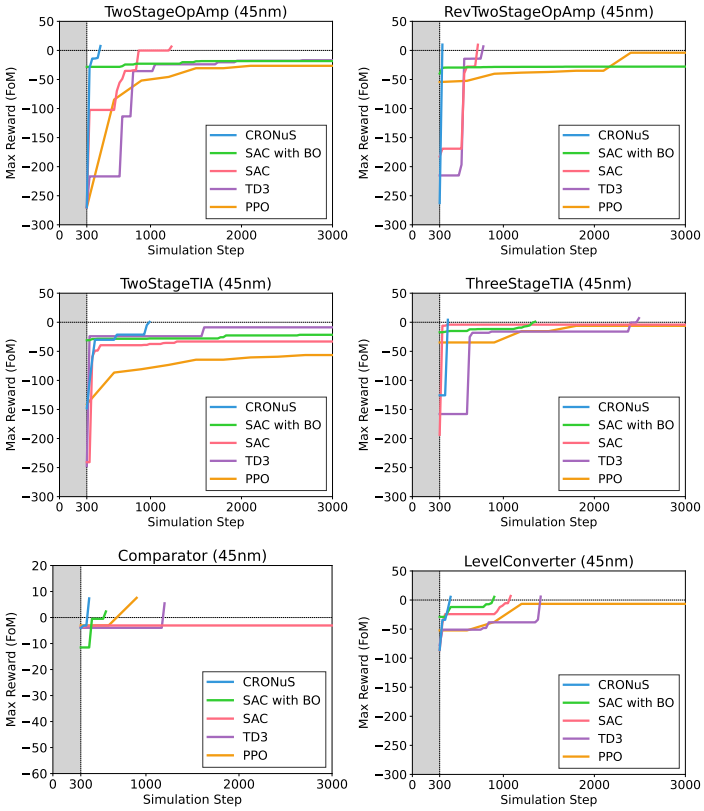


Fig. 5. Learning curves under fixed target specifications. Each algorithm is terminated early if the maximum reward (FoM), defined in Eq. (2), exceeds 0. The gray zone (300 steps) denotes the exploration phase. The solid lines and the shaded areas in all experimental results show the mean and standard deviations of five trials with randomly generated seeds.

DRL methods. The remarkable performance of this algorithm is largely attributed to its neural simulator. This simulator allows the policy network to be updated 20 times for each simulation step using a large amount of both real and synthetic data, as seen in Table I. Other algorithms without the neural simulator

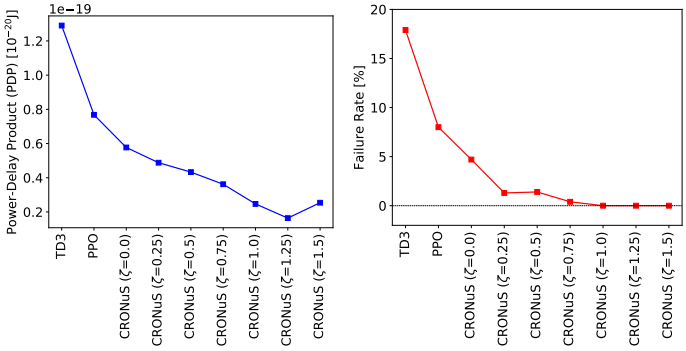


Fig. 6. Impact of confidence coefficient ζ in Comparator (45nm).

cannot replicate this. Note that PPO, which is used in [6], was unable to achieve generalizability before 4,500 steps in most cases, while TD3, an extension of DDPG [15], which was used in previous studies [4, 5, 8], had unstable training performance.

C. Robustness Evaluations

We show that the neural simulator is beneficial in obtaining a reliable policy in the presence of local process variation. On the Comparator (45nm) for interpretability, we compare CRONuS with PPO and TD3 for different confidence coefficient (ζ) , since they were effectively trained in 4,500 simulation steps, as seen in Fig. 4. To evaluate robustness, we take into account a noisy simulator based on Pelgrom's law [24] and assess performance based on circuit parameters found by each algorithm. Fig. 6 shows the impact of ζ on performance, power-delay product (PDP), and failure rate. Algorithms without conservative rewards have worse performance and more failure cases. Furthermore, CRONuS with $\zeta < 1.0$ still has few failure cases because it underestimates the impact of local process variation. CRONuS with $\zeta \geq 1.0$ can completely prevent failure cases. Note that a higher ζ does not necessarily mean a

worse performance. It might be the case where reducing delay contributes more to a smaller PDP than reducing power.

V. Conclusion

We proposed CRONuS with a neural simulator as a surrogate. It can be used to extend the data set and design a new conservative reward for rapid and stable policy training. The surrogate significantly improves sample efficiency by reducing the number of simulation runs, regardless of the type of circuits. The sample efficiency of CRONuS was five times better than that of other algorithms under varying target specifications. Furthermore, CRONuS is capable of designing analog circuits in the presence of local process variations.

References

[1] A. Potapov and S. Rodionov, "Genetic algorithms with dnn-based trainable crossover as an example of partial specialization of general search," in *International Conference on Artificial General Intelligence*. Springer, 2017, pp. 101–111.

[2] P. P. Kumar, K. Duraiswamy, and A. J. Anand, "An optimized device sizing of analog circuits using genetic algorithm," *European Journal of Scientific Research*, vol. 69, no. 3, pp. 441–448, 2012.

[3] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[4] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," *arXiv preprint arXiv:1812.02734*, 2018.

[5] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[6] 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)*. IEEE, 2020, pp. 490–495.

[7] Y. Li, Y. Lin, M. Madhusudan, A. Sharma, S. Sapatnekar, R. Harjani, and J. Hu, "A circuit attention network-based actor-critic learning approach to robust analog transistor sizing," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2021, pp. 1–6.

[8] W. Shi, H. Wang, J. Gu, M. Liu, D. Z. Pan, N. Sun *et al.*, "Rodesigner: Variation-aware optimization for robust analog design with multi-task rl," 2021.

[9] C. Minjeong, C. Youngchang, L. Kyongsu, and K. Seokhyeong, "Reinforcement learning-based analog circuit optimizer using g_m/id for sizing," in *Design Automation Conference*, 2023.

[10] G. Jian, C. Weidong, and Z. Xuan, "Rose: Robust analog circuit parameter optimization with sampling-efficient reinforcement learning," in *Design Automation Conference*, 2023.

[11] Z. Jinxin, B. Jiarui, H. Zhangcheng, Z. Xuan, and L. Ye, "Automated design of complex analog circuits with multiagent based reinforcement learning," in *Design Automation Conference*, 2023.

[12] P. Phuoc and C. Jaeyong, "Agd: A learning-based optimization framework for eda and its application to gate sizing," in *Design Automation Conference*, 2023.

[13] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[14] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[17] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[19] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[20] Ł. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model based reinforcement learning for atari," in *International Conference on Learning Representations*, 2019.

[21] W. Lee and F. A. Oliehoek, "Analog circuit design with dyna-style reinforcement learning," *arXiv preprint arXiv:2011.07665*, 2020.

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[23] W. Cao, M. Benosman, X. Zhang, and R. Ma, "Domain knowledge-infused deep learning for automated analog/radio-frequency circuit parameter optimization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1015–1020.

[24] A. Sheikholeslami, "Process variation and pelgrom's law [circuit intuitions]," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 1, pp. 8–9, 2015.