Design and Optimization of Low-Dropout Voltage Regulator Using Relational Graph Neural Network and Reinforcement Learning in Open-Source SKY130 Process

Zonghao Li, Anthony Chan Carusone
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
{zonghao.li, tony.chan.carusone}@isl.utoronto.ca

***Abstract*—Design automation and optimization for analog integrated circuits (ICs) are challenging, especially for transistor sizing. Given certain design specifications and circuit topology, circuit designers need to size various components to achieve the desired performance, possibly involving many optimization iterations. Recently, reinforcement learning (RL) has been applied to optimize analog circuits. The trained RL agents can achieve very high sample efficiency over evolutionary-based algorithms. By using the ability of transfer learning, the trained agent can be applied to optimize the same circuit across different technology nodes and even the circuits with different topologies. However, a significant bottleneck in applying machine learning (ML) techniques to analog IC design is the non-disclosure agreement (NDA) of the process development kit (PDK), which makes reproducibility of the prior art a big challenge. This work presents an RL framework that leverages the open-source SKY130 PDK to address the limitation above. We apply a novel heterogeneous graph neural network (GNN) called relational graph convolutional network (RGCN) as the function approximator of RL to capture more topological information about a circuit. As a proof-of-concept, low-dropout voltage regulators (LDO) are optimized by our proposed RL circuit optimizer framework to show its feasibility, achieving promising results.**

***Index Terms*—analog circuit, electronic design automation (EDA), graph neural network (GNN), low-dropout voltage regulator (LDO), machine learning (ML), open-source, optimization, reinforcement learning (RL).**

I. Introduction

Sizing transistors and other circuit components for a given circuit topology to meet desired specifications is a common task in analog integrated circuit (IC) design. This is a challenging problem since the design space can be huge, and simultaneously satisfying many specifications may involve many optimization iterations. Therefore, delivering a general automated methodology to optimize analog IC performance has always been a popular research topic [1]–[3]. Traditionally, evolutionary techniques such as genetic algorithms (GA) and particle swarm optimization (PSO) are applied to optimize analog circuits [4], [5], but they can be very sample-inefficient [6], [7]. Bayesian optimization (BO) is another black-box method to optimize analog circuits [1], [8]. However, a major drawback of BO is its scalability since its time complexity is cubic to the number of samples [9], [10]. Recently, reinforcement learning (RL) has been introduced as an alternative approach to optimize analog circuits [7], [9], [11]–[14]. Compared to evolutionary algorithms, the trained RL agent shows a big sample efficiency improvement [7]. In addition, the ability of transfer learning makes RL a possible candidate to reduce the time overhead of repetitive designs, which evolutionary algorithms and BO do not possess. In [7], the RL agent was trained with pre-layout simulation results. The trained agent was then applied to optimize the circuit in the post-layout simulations. In [11], the trained RL agent was used to optimize a trans-impedance amplifier (TIA) in different technology nodes and with different topologies. With these advantageous features of RL, it has attracted lots of research attention in recent years.

However, one of the biggest bottlenecks limiting progress on and the application of machine learning (ML) techniques to analog IC design is the non-disclosure agreement (NDA) required for access to a commercial process development kit (PDK). The NDA does not allow the computer code of detailed ML implementations in all papers mentioned above to be available in the public domain. Notable exceptions are [7] [15], which provide their implementations based on a simplified dummy PDK; thus, the designs cannot be fabricated. This hugely affects the reproducibility and reusability of the prior art and tremendously decelerates the research progress in this community. The need for an open-source ecosystem for ML methods to solve analog IC design questions on commercial PDKs is urgent. Google's initiative to provide fully open-source PDKs commenced in May 2020, including the SKY130 process from SkyWater. This work uses SKY130 to demonstrate our proposed framework because it is more mature and has a larger community than other open-source PDKs.

In this paper, we present a fully open-sourced RL framework

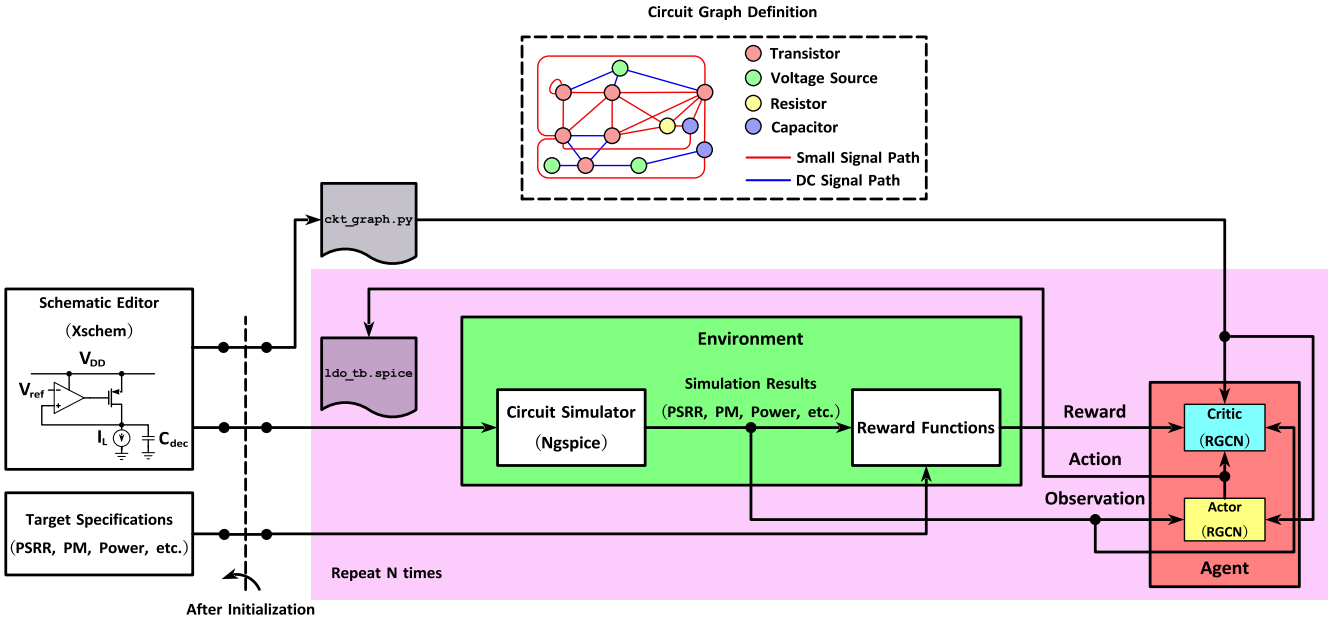


Fig. 1: Framework overview. Users start by defining the circuit schematic and target specifications. After the initialization, RL starts to optimize the circuit and improves the reward. Each iteration is one step, highlighted in pink.

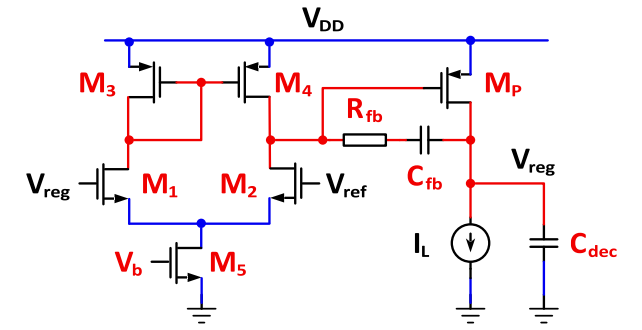
to optimize analog circuits in SKY130 process[1]. Similar to [11] [12], our RL agent is powered by the graph neural network (GNN) as the function approximator to capture the topological information of a circuit. This is because of a simple fact: a circuit is also a graph. It has its nodes (circuit components) and edge (wires). In particular, we introduce using the relational graph convolutional network (RGCN) to model the circuit topology. Compared to the homogeneous GNN used in [11] [12], RGCN is heterogeneous, which allows assigning each edge connection a type that has its learnable weights [16]. This helps in modeling the graph information of a circuit because not all wire connections are the same. For instance, in a circuit, we have wire connections for the DC biasing to ensure all transistors are biased at the appropriate operating point. There are wires designated for the small signals, and DC biasing nets are treated as the virtual ground to them. Based on this intuition, we can annotate the edge type of a circuit graph. Our simulation results show that with the application of this simple domain knowledge, better optimization results can be achieved. To demonstrate the feasibility of our proposition, we use our framework to optimize LDO circuits because it is a frequently used analog building block. Optimizing LDO circuits in the prior art primarily focuses on using GA [17] [18] and equation-based methods [19]. Using RL to optimize LDO was briefly mentioned in [11] and [13] but still missing many details. For the proof-of-concept, our RL framework is used to optimize two LDO circuits with some hypothetical specifications. The structure of this paper is organized as follows: Section II will present the overview of our framework, Section III will show the experimental results using our framework, and Section IV will conclude the paper.

[1]Source code: https://github.com/ChrisZonghaoLi/sky130_ldo_rl

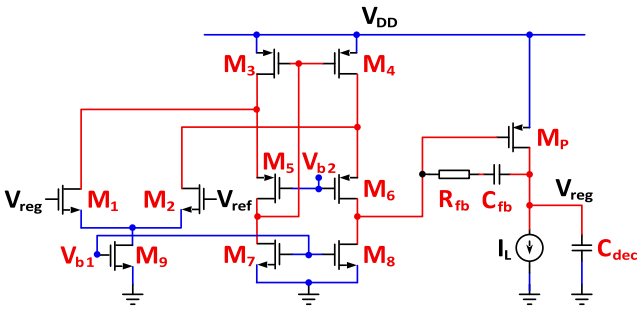
II. Overview

A. Framework Overview

Fig. 1 shows the overview of our proposed framework. The optimization task of an analog IC starts with the desired applications, specifications, and circuit topologies. In this work, we choose LDO circuits as the optimization candidates. The open-source schematic capture program Xschem [20] is used to export the netlist of the LDO circuit `ldo_tb.spice`. In this netlist, all circuit component parameters, such as the length and width of transistors, are all parameterized so that the actions from the RL agent can modify them. The graph description defines the node and edge connections of a circuit, which is needed for a GNN. One can manually define them based on the circuit schematic. This information is housed in `ckt_graph.py`. Next, the circuit netlist will be simulated with Ngspice. The simulation results are passed to the reward functions to calculate the reward. The attributes of each circuit component after the DC operating point (`OP`) analysis are stored and serve as the observations of the RL agent. The RL agent is a typical actor-critic structure [21], inside which the neural network (NN) architectures are dependent on the circuit graph definition provided by `ckt_graph.py`. The actor will take the observation as the input and generate the output action for the next iteration, which is then fed to `ldo_tb.spice` and updates the netlist. The critic takes observation and action as the input to return the corresponding expectation of the long-term reward. Once this iteration is done, the same behavior repeats for N times where N is the user-defined step number.



(a)



(b)

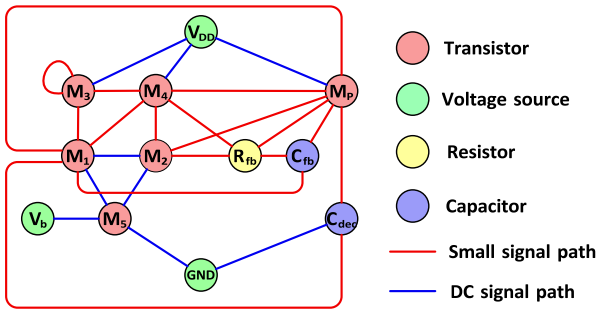
Fig. 2: (a) LDO1, where its EA is a simple diff-pair. (b) LDO2, where its EA is a folded cascode amplifier. The parameters of the components in red will be adjusted by RL during the optimization. The nets in red are primarily for AC signals, and the nets in blue are primarily for DC biasing.

TABLE I: LDO Specifications

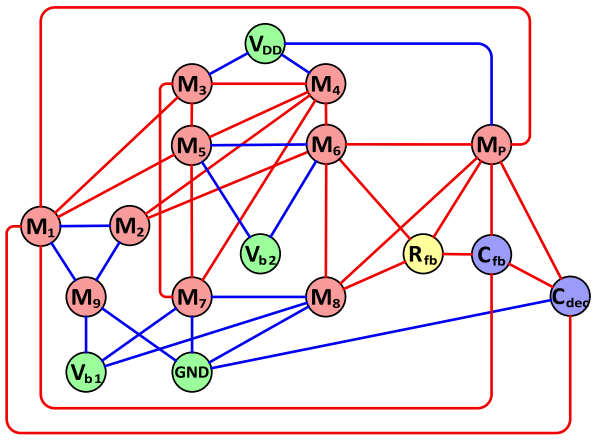
$V_{DD}\,(V)$	2
$V_{reg}\,(V)$	1.8
$V_{drop}\,(mV)$	≤ 200
I_L	$[10\,\mu A, 10\,mA]$
$PSRR_{\leq 10kHz}\,(dB)$	≤ -30 (at $I_{L,min}$ and $I_{L,max}$)
$PSRR_{\leq 1MHz}\,(dB)$	≤ -20 (at $I_{L,min}$ and $I_{L,max}$)
$PSRR_{\geq 1MHz}\,(dB)$	≤ -5 (at $I_{L,min}$ and $I_{L,max}$)
$PM\,(deg)$	$\geq 60°$ (at $I_{L,min}$ and $I_{L,max}$)
$I_q\,(\mu A)$	≤ 200 (≤ 400 for LDO2)
$\Delta V_{reg}\,(mV)$	≤ 36
C_{dec}	As small as possible

B. LDO Circuits

In this paper, we consider two simple LDO circuits to illustrate the workflow of our proposed framework. Fig. 2 shows the schematic of two LDO circuits. The first one is a vanilla LDO consisting of a diff-pair error amplifier (EA) and a PMOS pass transistor. The second one has a similar architecture, except the EA is folded cascode. For simplicity, we do not consider the non-ideality of the voltage source V_{ref}, and we tie the output V_{reg} directly to the positive terminal of the EA. In both LDOs, a decoupling capacitor C_{dec} filters out high-frequency noise. We set $V_{DD} = 2\,V$ and $V_{ref} = 1.8\,V$. The RL will tweak the circuit components highlighted in red during the optimization. Table I shows the desired specifications for LDO circuits. Noted



(a)



(b)

Fig. 3: (a) LDO1, where its EA is a simple diff-pair. (b) LDO2, where its EA is a folded cascode amplifier. The nets in red are primarily for AC signals, and the nets in blue are primarily for DC biasing.

that first, these specifications are hypothetical and are not guaranteed to be all achievable. Second, the $PSRR$ and PM specifications are posed at both $I_{L,min}$ and $I_{L,max}$ as the load conditions change their performance [22]. Third, optimizing the decoupling capacitor C_{dec} should take place only when other hard specifications are satisfied.

C. RGCN

Converting a circuit netlist to a graph may retain more useful topological information during the learning process. In [11], the authors used graph convolutional neural network (GCN) as the function approximator for the RL. They demonstrated its superior performance and transfer learning ability. One advantage of GCN is its node-embedding ability, allowing it to take each node's attributes and aggregate this information with its neighboring nodes [23]. Similarly, authors in [12] used graph attention net (GAT) to model the circuits, achieving better rewards.

A typical graph consists of two fundamental parts: nodes and edges. In a circuit schematic, there exist different types of nets. For example, some nets are primarily DC signal paths, and others are small signal AC paths. Inspired by

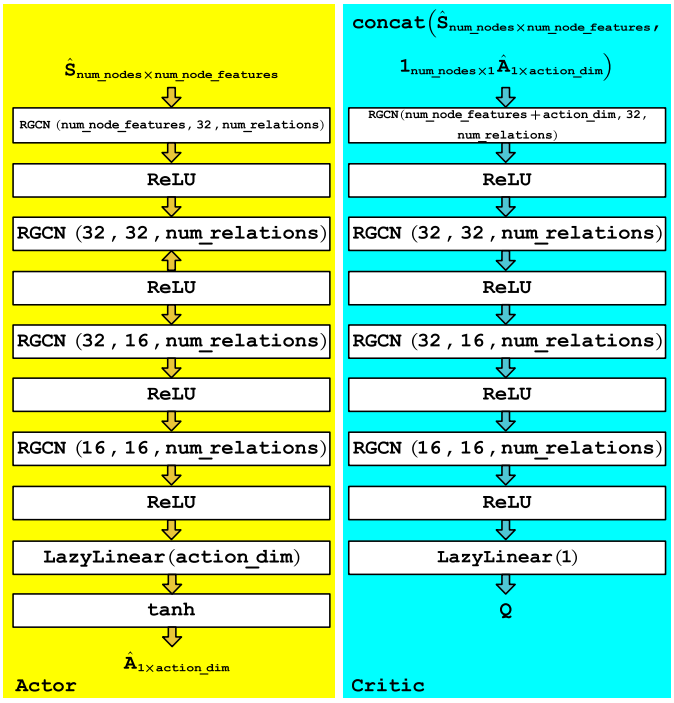


Fig. 4: RGCN architecture for actor and critic. Depending on the output, the only differences are the output layers. $\hat{S}$ is the normalized observation from the environment, $\hat{A}$ is the action predicted by the actor before denormalization, and Q is the predicted value for the value function.

this observation, we applied a heterogeneous graph here to model the circuit allowing edge types to be differentiated. In particular, we use RGCN in this work. RGCN can be formulated as [16]:

$$h_i^{l+1} = \sigma\Big(\sum_{r\in R}\sum_{j\in N_i^r} c_{i,r}^{-1} W_r^{(l)} h_j^{(l)} + W_0^l h_i^l\Big) \tag{1}$$

A significant difference in RGCN compared to GCN and GAT is the introduction of different edge types r, where each edge type has an independent learnable weight matrix W_r^l for layer l. Additionally, an extra W_0^l matrix is designated for the self-connection node's edge. N_i^r indicates that node i is connected with edge type r. Therefore, we use RGCN as our actor and critic networks to better preserve the circuit graph knowledge. We use two separate RGCN networks for actor and critic, whose architectures are identical except for the first and last few layers, shown in Fig. 4.

D. RL Formulations

In our framework, the optimization is done with RL. Specifically, we use deep deterministic policy gradient (DDPG) [21], [24] as our RL algorithm due to the following reasons. First, its implementation is straightforward. Second, it is an off-policy algorithm, which means the training dataset can be collected offline. Since the SKY130 PDK is open-source, one can prepare a massive standalone training dataset for a circuit, and this training dataset can be used as the replay buffer to train the corresponding DDPG agent. Third, we treat the component sizes as a continuous design space rather than discrete as in [7] [12] to reduce the dimension of action space [11]. Forth, it can be more sample efficient than on-policy algorithms [25].

TABLE II: LDO Action Space

LDO1	$W\,(\mu m)$	$L\,(\mu m)$	M	V
M_1 - M_5	$[1, 100]$	$[0.5, 2]$	1	
M_P	$[10, 100]$	$[0.5, 1]$	$[100, 2000]$	
R_{fb}	0.35	1	[1] $[1, 20]$	
C_{fb}	10	10	[2] $[1, 50]$	
C_{dec}	30	30	[3] $[10, 300]$	
V_b				$[0.9, 1.4]$
LDO2	$W\,(\mu m)$	$L\,(\mu m)$	M	V
M_1 - M_9	$[1, 100]$	$[0.5, 2]$	1	
M_P	$[10, 100]$	$[0.5, 1]$	$[100, 2000]$	
R_{fb}	0.35	1	[1] $[1, 20]$	
C_{fb}	10	10	[2] $[1, 50]$	
C_{dec}	30	30	[3] $[10, 300]$	
V_{b1}				$[0.9, 1.4]$
V_{b2}				$[0, 1]$

[1] Corresponds to $[476.7, 9335]\,\Omega$.
[2] Corresponds to $[0.2076, 10.38]\,pF$.
[3] Corresponds to $[18.23, 546.8]\,pF$.

1) Action: Each circuit component has its own action space, which the RL agent will explore during the optimization. Table III summarizes the action range for each circuit component. We use high sheet resistance poly resistor for R_{fb} with a sheet resistance of 1112.4 $\Omega/\square$. To simplify the design space, we fix its $W = 0.35\,\mu m$ and $L = 1\,\mu m$ and let the device multiplier M be the only action variable. Similarly, we use the metal-insulator-metal (MIM) capacitor for C_{fb} and C_{dec} which has a capacitance density of 2.2 $fF/\mu m^2$. We constrain their W and L to make their M the only action variables. We further notice that due to the symmetry of the EA, the action variables can be further deduced. For instance, M_1 from the EA in LDO1 from Fig. 2 shares the same size as M_2, and M_3 shares the same size as M_4. Therefore, we can just optimize M_1 and M_3. Unlike prior art that fixed the basing conditions for circuits [7], [9], [11], [12], we also make the basing voltage as action variables, as shown in III. This is because the biasing conditions strongly affect the circuit performance.

As an example, the actor's predicted action for LDO1 in Fig. 2(a) is $\hat{A}$ = [$\hat{W}_1$, $\hat{L}_1$, $\hat{W}_3$, $\hat{L}_3$, $\hat{W}_5$, $\hat{L}_5$, $\hat{W}_P$, $\hat{L}_P$, $\hat{M}_P$, $\hat{M}_{R_{fb}}$, $\hat{M}_{C_{fb}}$, $\hat{M}_{C_L}$, $\hat{V}_b$]. Note that in Fig. 4, the output nonlinear function of the actor `tanh` confines the output action to be within $[-1, 1]$. To help the DDPG agent explore the action space, we further apply the truncated uniform noise $\mathcal{N}$ bounded by $[-1, 1]$ and add it to $\hat{A}$. It has an exponential decay factor $\alpha = 0.999$ for LDO1 and 0.9995 for LDO2, and its initial noise volume $\sigma = 2$. After the noise addition, we again clip $\hat{A}$ and ensure its range is enclosed within $[-1, 1]$. Since each entry of $\hat{A}$ is well-defined, the following min-max denormalization is used to restore the action to the original scale so that Ngspice can simulate them:

$$A = \hat{A} \odot \left(\frac{A_{max} - A_{min}}{2}\right) + \left(\frac{A_{max} + A_{min}}{2}\right) \quad (2)$$

2) Reward: The design of the reward function significantly impacts the performance of RL in general. The reward function can be very sparse in many scenarios, such as robotic control and video game playing. Reward sparsity is one of the biggest challenges in RL [26]. Careful reward engineering can reduce the reward sparsity and thereby significantly improve the sample efficiency and robustness of RL. This requires a solid understanding of domain knowledge. Similar to the prior art, the following potential-based reward function was used [7], [11], [12], [15] in our framework:

$$r_i = w_i \beta_i \left\{ \frac{O_i - O_i^*}{O_i + O_i^*}, 0 \right\}_{min} \quad (3)$$

$$R = \begin{cases} \sum_{i=1}^{N} r_i & if\ \exists r_i < 0 \\ \sum_{j=1}^{K} w_j \beta_j \frac{O_j - O_j^*}{O_j + O_j^*} + C & else \end{cases} \quad (4)$$

where the simulation results are denoted as O, and the desired specifications are O^*. Particularly, we define two different types of specifications: hard specifications denoted as O_i^* in Eq. 3 and soft specifications denoted as O_j^* in Eq. 4. Hard specifications should be met unconditionally, and they are usually defined to the designers with some exact numbers. Soft specifications often do not have concrete numerical objectives, and they should be optimized only after all hard specifications are satisfied. We can arbitrarily pick a reasonable value for O_j^* as a hypothetical goal. As an example, for the LDO specifications considered in Table I, the area of C_{dec} is the soft specification. We can pick its O_j^* as the smallest C_{dec} given its action range defined in Table III. It will only be optimized when all other hard specifications are met. Depending on whether the optimization task for a specification is a minimization or a maximization job, β will be either -1 or 1, respectively. Finally, we can add a constant C to R in Eq. 4 to reward the optimization success of satisfying all hard specifications.

As shown in Eq. 3, the reward for each hard specification will be bounded by $[-1, 0]$. Once a hard specification is met, no extra score will be granted to that specification to avoid over-designing. w_i controls the weight of i^{th} hard specification in the reward function. In the prior art, w was assumed to be 1 across all specifications. We argue that this simplification may lead to worse sample efficiency and may even converge to some unwanted design choices. For instance, imagine for the given LDO specifications in Table I, if one assigns $w = 1$ for both $PSRR_{\leq 10kHz}$ and $PSRR_{\leq 1MHz}$, then the RL will treat optimizing these two specs equally. However, having a specification of $PSRR_{\leq 1MHz} \leq -20\,dB$ indirectly implies that $PSRR_{\leq 10kHz}$ will be also smaller than $-20\,dB$. In addition, it is easier to improve the low-frequency $PSRR$ performance compared to high-frequency $PSRR$. Therefore, it is highly possible that more design candidates meet the $PSRR_{\leq 10kHz}$ specification during the optimization process but violate the $PSRR_{\leq 1MHz}$ requirement. Having the same w for these two might potentially lead the RL optimizer to gradually give up optimizing $PSRR_{\leq 1MHz}$ as optimizing $PSRR_{\leq 10kHz}$ has a higher cumulative reward. Another reason is that having very high $PSRR$ around low frequency may not be necessary as most noises will appear at the MHz-GHz range [27]. A similar argument can be posed to the phase margin PM. It is possible to have a design candidate whose $PSRR$ performance is promising, but its PM at $I_{L,min}$ is poor, potentially leading to instability. It will be hard to rule out these unqualified designs if w of PM is the same as other specifications.

To tackle these concerns, we propose first letting all $w = 1$ except $w_{PSRR_{\leq 10kHz}} = 0.5$, and applying the following modifications to the $PSRR$ and PM reward functions:

$$r_{PSRR} = \begin{cases} -1 & if\ PSRR \geq 0\,dB \\ -w_i \left\{ \frac{O_i - O_i^*}{O_i + O_i^*}, 0 \right\}_{min} & else \end{cases} \quad (5)$$

$$r_{PM} = \begin{cases} -1 + w_i \left\{ \frac{O_i - O_i^*}{O_i + O_i^*}, 0 \right\}_{min} & if\ PM \leq 45^\circ \\ w_i \left\{ \frac{O_i - O_i^*}{O_i + O_i^*}, 0 \right\}_{min} & else \end{cases} \quad (6)$$

We see an extra -1 penalty posed for both $PSRR$ and PM reward functions. For $PSRR$, we do not want it to be larger than $0\,dB$ at any frequency point since it will amplify the noise. We also want LDOs to be very stable under different load conditions, so we give an extra punishment to designs whose PM are smaller than 45°.

3) Observation: As shown in Fig. 1, the observations from the environment are the simulation results of the circuits. Particularly, they are the internal device parameters obtained by the `OP` analysis. For example, we use the vector S_M = [i_d, g_m, g_{ds}, V_{th}, V_{dsat}, V_{ds}, V_{gs}] to represent the observation from a transistor. The internal device parameters for the resistor, capacitor, and voltage source are their resistance, capacitance, and voltage value, respectively. We use zero padding to ensure all vectors have the same dimension. Finally, we stack these vectors to form our observation matrix S. As an example, the following matrix represents the observation for LDO1:

$$S = \begin{bmatrix} S_{M_N} & \mathbf{0}_{6\times 6} \\ \mathbf{0}_{6\times 7} & diag(V_b, V_{DD}, V_{GND}, R_{fb}, C_{fb}, C_L) \end{bmatrix} \quad (7)$$

$$S_{M_N} = \begin{bmatrix} S_{M_1} \\ S_{M_2} \\ \vdots \\ S_{M_P} \end{bmatrix} \quad (8)$$

It is important to normalize these observations as the performance of RL is sensitive to the scale of the input features. However, a problem for S_{M_N} in Eq. 8 is that the

Algorithm 1 Find $\mu_{S_{M_N}}$ and $\sigma_{S_{M_N}}$

Require: $n \in \mathbb{N}^+$ ▷ Such as $n = 100$
Require: $S^* = [\,]$ ▷ Empty list to store observations
$i \leftarrow 0$
while $i < n$ **do**
  Sample A randomly from Table III
  Run OP analysis
  Store S_{M_N} in S^*
  $i \leftarrow i + 1$
end while
$\mu_{S_{M_N}} = mean(S^*)$
$\sigma_{S_{M_N}} = std(S^*)$

transistor's internal device parameters are unlike other passive devices and voltage sources that have a clear bounded range. This prohibits using min-max normalization defined in Eq. 2. Therefore, we use the common z-score normalization:

$$\hat{S}_{M_N} = \frac{S_{M_N} - \mu_{S_{M_N}}}{\sigma_{S_{M_N}}} \tag{9}$$

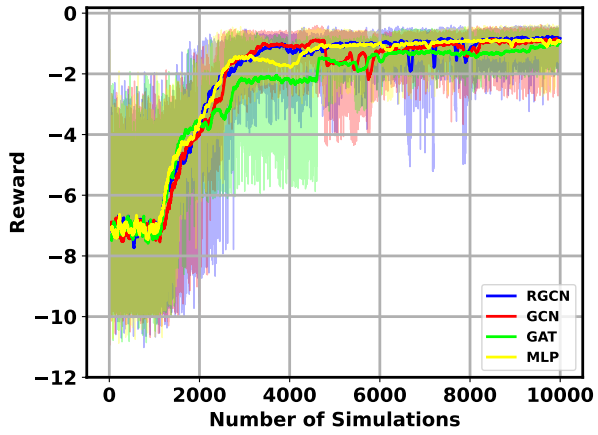
where $\mu_{S_{M_N}}$ and $\sigma_{S_{M_N}}$ are the mean and standard deviation of transistor internal device parameters. In [11], the authors find $\mu_{S_{M_N}}$ and $\sigma_{S_{M_N}}$ across all transistors in each simulation. However, this approach may have a few issues. First, if $\mu_{S_{M_N}}$ and $\sigma_{S_{M_N}}$ change from one simulation to another, then even identical $\hat{S}_{M_N}$ from different simulations may represent very distinct observations S_{M_N}. Second, this method does not work well when a circuit does not have many transistors. Imagine an extreme case with a single-transistor common-source amplifier, and the above approach is not applicable. To solve these two possible problems, we propose to simply conduct the OP analysis of the entire circuit for a few runs at first by randomly sampling the action space. We then calculate the $\mu_{S_{M_N}}$ and $\sigma_{S_{M_N}}$ from all of these simulation results and use them to normalize S_{M_N}. For example, in this work, we run 100 OP simulations for both LDO1 and LDO2. We further clip $\hat{S}_{M_N}$ to be bounded by $[-5, 5]$. This is summarized in Algorithm 1.

III. Experiments

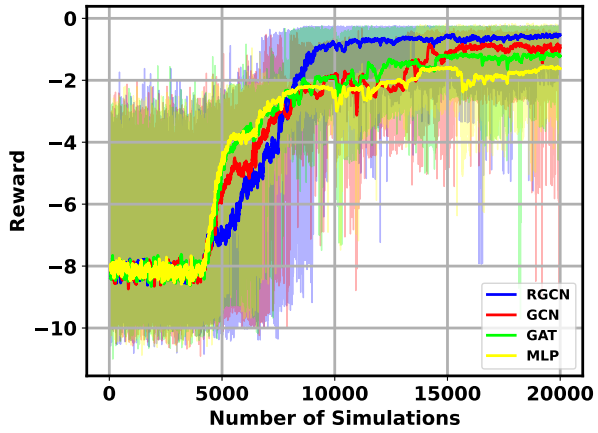
The detailed implementation of our framework is given in Algorithm 2. To investigate how RGCN affects RL performance in LDO optimizations, we have also run three other benchmark experiments: GCN+DDPG [11], [12], GAT+DDPG [12], and multi-layer perceptron (MLP)+DDPG [7], [15]. To have a fair comparison, each of these experiments has been conducted independently three times, and all NNs have the same architecture as in Fig. 4. We implement all our NNs using PyG [28], and our LDO environments are compatible with Gym [29]. Fig. 5 shows the plot of reward versus the number of simulations t of LDO1 and LDO2. It takes about 12 hours to run 10000 simulations for LDO1 and 24 hours to run 20000 simulations for LDO2. To help DDPG search for the optimal policy faster, we apply 1000 and 4000 initial random exploration steps for LDO1 and LDO2, respectively. The episode length w is 50, the size of our memory buffer M is 100000, and the batch size B is 128. For the uniform noise model $\mathcal{N}$, we use an exponential decay factor $\alpha = 0.999$ for LDO1 and 0.9995 for LDO2, and its initial noise volume $\sigma = 2$.

Algorithm 2 RL circuit optimizer

Require: Memory buffer M, step number $N \in \mathbb{N}^+$, episode length $w \in \mathbb{N}^+$, initial random step $n \in \mathbb{N}^+$, batch size $B \in \mathbb{N}^+$, initial actor NN parameter θ, and initial critic NN parameters ϕ.
1: **if** $\mu_{S_{M_N}} = \emptyset$ or $\sigma_{S_{M_N}} = \emptyset$ **then**
2: Run Algorithm 1
3: **end if**
4: $t \leftarrow 0$
5: $\theta_{tar} \leftarrow \phi$, $\phi_{tar} \leftarrow \phi$ ▷ Initial target actor and critic NN
6: $\gamma \leftarrow 0.99$, $\rho \leftarrow 0.005$ ▷ DDPG hyperparameters
7: Initial circuit with A and S
8: Initial uniform truncated noise $\mathcal{N}$ and its σ and α
9: **while** $t < n$ **do** ▷ Random exploration steps
10: Sample A randomly from Table III
11: Run the environment
12: Normalize S and A according to Eq. 9 and Eq. 2
13: $\hat{S} \leftarrow clip(\hat{S}, -5, 5)$
14: $d \leftarrow False$ ▷ Done signal
15: **if** $R \geq 0$ **then**
16: $d \leftarrow True$
17: Run line 7 ▷ Reset the environment
18: **end if**
19: Store $(\hat{S}, \hat{A}, R, \hat{S}', d)$ to M ▷ $\hat{S}'$ is the next obs.
20: $t \leftarrow t + 1$
21: **end while**
22: **while** $t < N$ **do**
23: $j \leftarrow 0$
24: **while** $j < w$ and $d = False$ **do**
25: From actor: $\hat{A} = clip(\pi_\theta(\hat{S}) + \mathcal{N}, -1, 1)$ ▷ π is the policy
26: $\alpha \leftarrow min\{\alpha\sigma, 0.1\}$
27: Denormalize $\hat{A}$ according to Eq. 2
28: Do lines 11-19
29: **if** $t \geq B$ **then**
30: Randomly sample B transitions from M
31: Compute target: $y(R, \hat{S}, d) = R + \gamma(1 - d)Q_{\phi_{tar}}(\hat{S}, \pi_{\theta_{tar}}(\hat{S}))$
32: Update critic: $\nabla_\phi MSE(Q_\phi(\hat{S}, \hat{A}), y(R, \hat{S}, d))$
33: Update actor: $\nabla_\theta \frac{1}{B} \sum(Q_\phi(\hat{S}, \pi_\theta(\hat{S}))$
34: Update target actor: $\theta_{tar} \leftarrow \rho\theta_{tar} + (1 - \rho)\theta$
35: Update target critic: $\phi_{tar} \leftarrow \rho\phi_{tar} + (1 - \rho)\phi$
36: **end if**
37: $t \leftarrow t + 1$, $j \leftarrow j + 1$
38: **end while** ▷ Complete an episode
39: Run line 7 ▷ Reset the environment
40: **end while**



(a)



(b)

Fig. 5: Reward versus the number of simulations with different NN as the DDPG function approximator for LDO1 (a) and LDO2 (b). All plots are generated from three independent runs.

Simulation results in Fig. 5(a) show that both RGCN and GCN converge faster (around $t = 3000$) than GAT and MLP (around $t = 5000$), but they all eventually converge to a similar reward value after around $t = 10000$. Since the final reward is not larger than 0 for any of these experiments according to Eq. 3, we conclude that not all specifications in Table I are met. Both GAT and MLP lose the improvement after around $t = 3000$ temporarily and recover the momentum when $t = 5000$. Unlike the observations made in [12], we did not see the superiority of GAT over GCN in our experiment. This is probably because we did not fine-tune the hyperparameters of GAT.

Fig. 5(b) shows the reward simulation results of LDO2. RGCN converges to the highest reward value compared to others, despite its improvement between $t = 5000$ and $t = 8000$ being the slowest. Again, none of the final reward values are larger than 0, so some specifications in Table I are violated. Unlike Fig. 5(a), in this experiment we see that GCN struggles to converge to a good reward value until around $t = 15000$, which is still lower than what RGCN has achieved. MLP, despite its performance in the LDO1 experiment, has the worst reward value among these four benchmark tests. We conclude that since LDO2 is more complicated than LDO1 in terms of action space and topology complexity, the advantage of RGCN may come into play. On the other hand, since homogenous NNs do not differentiate different edge types, their expressive power might be limited when the graphs they need to represent become complicated.

TABLE III: Optimized LDO Circuit Component Values

LDO1 ($R = -0.39$)	$W\ (\mu m)$	$L\ (\mu m)$	M	V
M_1 and M_2	13	2	1	
M_3 and M_4	80	0.5	1	
M_5	14.27	1.22	1	
M_P	63.65	0.5	142	
R_{fb}	0.35	1	1	
C_{fb}	10	10	4	
C_{dec}	30	30	262	
V_b				1.37
LDO2 ($R = -0.23$)	$W\ (\mu m)$	$L\ (\mu m)$	M	V
M_1 and M_2	12.3	1.77	1	
M_3 and M_4	37	0.55	1	
M_5 and M_6	12.33	0.5	1	
M_7 and M_8	91.4	1.7	1	
M_9	32.27	0.56	1	
M_P	11.55	0.5	275	
R_{fb}	0.35	1	1	
C_{fb}	10	10	9	
C_{dec}	30	30	239	
V_{b1}				1
V_{b2}				0.025

For validation purposes, we show the Ngspice simulation results of the optimized LDO2 in Fig. 6. Noted that we did not include any line regulation test in our simulations or reward function to reduce the runtime. We present the line regulation simulation results here just for the completeness of validation. Table IV summarizes the optimized LDO performance compared to the specifications in Table I. The numbers in bold are the results that violate the specifications. For LDO1, we see some pretty mild violations of $PSRR_{\leq 10kHz}$ and PM at $I_{L,min}$, as well as ΔV_{reg}. We also notice that the C_{dec} area is not optimized yet since not all hard specifications are met. The optimizer finds difficulty in optimizing $PSRR_{\geq 1MHz}$ at $I_{L,max}$ and fails to meet the specification. The intuition behind this can be explained as follows: to obtain a fairly good PM, it is necessary to have enough separation between the dominant and second poles of the LDO. However, the poles will be moving with the change of the load current I_L. Particularly, in our case, the dominant pole will move toward the lower frequency, and the second pole will move toward the higher frequency when I_L increases, leading to a better PM but a worse $PSRR$ at medium-high frequency [22]. Therefore, a trade-off must be made between $PSRR$ and PM here. By using a folded cascode EA and allowing a higher power budget, we see some obvious improvements from LDO2 versus LDO1. However, the specification on $PSRR_{\geq 1MHz}$ has not been attained either. This challenge might be solved

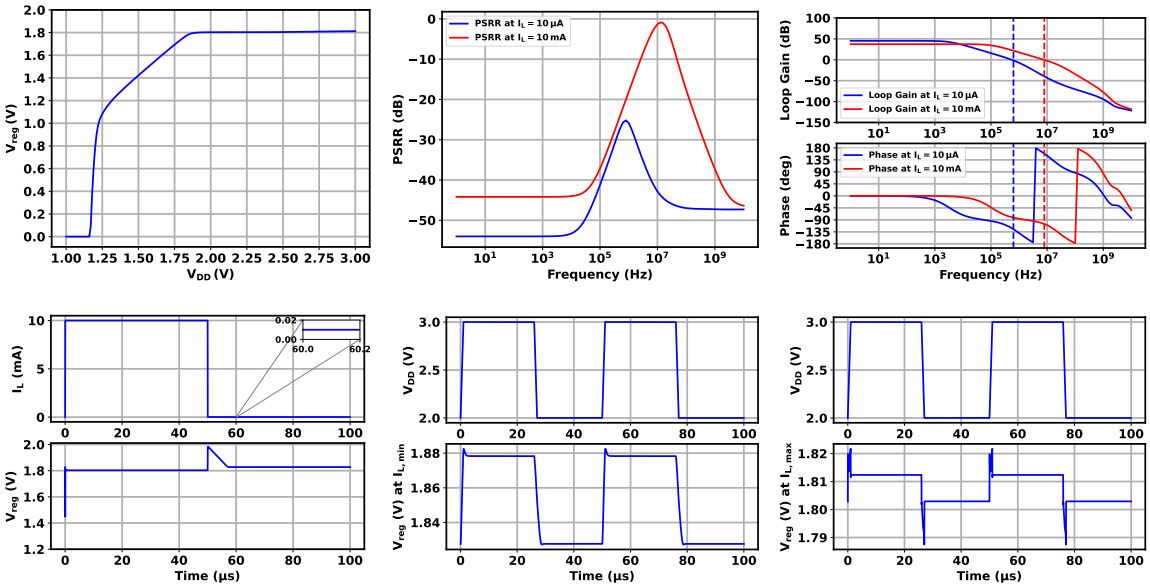


Fig. 6: Simulation results of the optimal LDO2. We also show its line regulation results in the last two subplots that are not included in our reward function.

TABLE IV: Optimization Results of LDOs

	Specifications	LDO1	LDO2
$V_{drop}\,(mV)$ at $I_{L,max}$	≤ 220	203	197
I_L	$[10\,\mu A, 10\,mA]$	$[10\,\mu A, 10\,mA]$	-53.53 at $I_{L,min}$ and -44.1 at $I_{L,max}$
$PSRR_{<10kHz}\,(dB)$	≤ -30 (at $I_{L,min}$ and $I_{L,max}$)	**-29.3** at $I_{L,min}$ and -34 at $I_{L,max}$	-25.2 at $I_{L,min}$ and -20 at $I_{L,max}$
$PSRR_{<1MHz}\,(dB)$	≤ -20 (at $I_{L,min}$ and $I_{L,max}$)	-22 at $I_{L,min}$ and -23.2 at $I_{L,max}$	-25.86 at $I_{L,min}$ and **-0.9** at $I_{L,max}$
$PSRR_{\geq 1MHz}\,(dB)$	≤ -5 (at $I_{L,min}$ and $I_{L,max}$)	-25.75 at $I_{L,min}$ and **-0.32** at $I_{L,max}$	61.2° at $I_{L,min}$ and 78.2° at $I_{L,max}$
$PM\,(deg)$	$\geq 60°$ (at $I_{L,min}$ and $I_{L,max}$)	**54.4**° at $I_{L,min}$ and 61° at $I_{L,max}$	398
$I_q\,(\mu A)$	≤ 200 (≤ 400 for LDO2)	175.32	24.74
$\Delta V_{reg}\,(mV)$	≤ 36	**40**	435
$C_{dec}\,(pF)$	As small as possible	476.78	

TABLE V: Comparison Table For Prior Analog Circuit Automation Works Using RL

	[9]	[15]	[11]	[12]	**This Work**
RL algorithm	DDPG	PPO	DDPG	PPO	DDPG
RL function approximator	RNN	MLP	GCN	GCN, GAT	RGCN
Circuits considered	TIA	TIA, OTA	OTA, TIA, LDO	OTA, PA	LDO
Technology	180nm CMOS	16nm CMOS	180nm CMOS	45nm CMOS, 150nm GaN	SKY130 CMOS
Action dimensions	19	29	25	15	18
Number of specifications	4	7	5	4	13
Fully open-sourced?	No	No	No	No	Yes

by seeking more advanced LDO topologies, such as the one in [30].

IV. Conclusion

A fully open-sourced RL analog circuit design automation framework is presented in this work. We suggest using RGCN as the function approximator for the RL. As a proof-of-concept, we demonstrate optimizing LDO circuits in the open-source SKY130 process with our proposed framework. Simulation results show that our RL optimizer can reach better results compared to using other homogeneous NNs as the function approximators, especially when the circuit topology is more complex. Table V tabulates the comparison between our framework and some other state-of-the-art RL analog circuit automation works, and ours is capable of optimizing a broad range of specifications. More importantly, our entire work is completely open-sourced, which hopefully can be reused and improved by other people to accelerate the research progress in this domain.

V. Acknowledgement

The authors would like to thank Jiacheng Yang and Jonas Guan from the University of Toronto and Hossein Shakiba from Huawei Technologies Canada for some very insightful discussions.

REFERENCES

[1] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3306–3314, PMLR, 10–15 Jul 2018.

[2] T. Liao and L. Zhang, "Parasitic-aware gp-based many-objective sizing methodology for analog and rf integrated circuits," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 475–480, 2017.

[3] B. Liu, F. V. Fernández, Q. Zhang, M. Pak, S. Sipahi, and G. Gielen, "An enhanced moea/d-de and its application to multiobjective analog cell sizing," in *IEEE Congress on Evolutionary Computation*, pp. 1–7, 2010.

[4] R. Zhou, P. Poechmueller, and Y. Wang, "An analog circuit design and optimization system with rule-guided genetic algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5182–5192, 2022.

[5] D. Joshi, S. Dash, U. Agarwal, R. Bhattacharjee, and G. Trivedi, "Analog circuit optimization based on hybrid particle swarm optimization," in *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 164–169, 2015.

[6] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović, "Bagnet: Berkeley analog generator with layout optimizer boosted with deep neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[7] 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)*, pp. 490–495, 2020.

[8] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou, "An efficient bayesian optimization approach for automated optimization of analog circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 6, pp. 1954–1967, 2018.

[9] H. Wang, J. Yang, H.-S. Lee, and S. Han, "Learning to design circuits," 2020.

[10] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams, "Scalable bayesian optimization using deep neural networks," 2015.

[11] 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)*, pp. 1–6, 2020.

[12] 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*, ACM, jul 2022.

[13] K.-E. Yang, C.-Y. Tsai, H.-H. Shen, C.-F. Chiang, F.-M. Tsai, C.-A. Wang, Y. Ting, C.-S. Yeh, and C.-T. Lai, "Trust-region method with deep reinforcement learning in analog design space exploration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, dec 2021.

[14] 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)*, pp. 1–6, 2021.

[15] K. Settaluri, Z. Liu, R. Khurana, A. Mirhaj, R. Jain, and B. Nikolic, "Automated design of analog circuits using reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 2794–2807, 2022.

[16] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017.

[17] J. Lopez-Arredondo, E. Tlelo-Cuautle, and R. Trejo-Guerra, "Optimizing an ldo voltage regulator by evolutionary algorithms considering tolerances of the circuit elements," in *2015 16th Latin-American Test Symposium (LATS)*, pp. 1–5, 2015.

[18] J. López-Arredondo, E. Tlelo-Cuautle, and F. V. Fernández, "Optimization of ldo voltage regulators by nsga-ii," in *2016 13th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, pp. 1–4, 2016.

[19] S. DasGupta and P. Mandal, "An automated design approach for cmos ldo regulators," in *2009 Asia and South Pacific Design Automation Conference*, pp. 510–515, 2009.

[20] S. Schippers, "Xschem." https://github.com/StefanSchippers/xschem, 1998.

[21] 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," 2019.

[22] R. J. Milliken, J. Silva-Martinez, and E. Sanchez-Sinencio, "Full on-chip cmos low-dropout voltage regulator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 9, pp. 1879–1890, 2007.

[23] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[24] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[26] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2018.

[27] D. Tonietto and A. C. Carusone, "Designcon 2018 chip-level power integrity methodology for high-speed serial links," 2017.

[28] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[30] M. El-Nozahi, A. Amer, J. Torres, K. Entesari, and E. Sanchez-Sinencio, "High psr low drop-out regulator with feed-forward ripple cancellation technique," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 3, pp. 565–577, 2010.