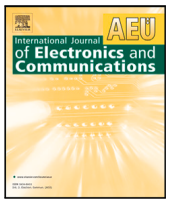
Int. J. Electron. Commun. (AEÜ) 192 (2025) 155697

Contents lists available at ScienceDirect

Int. J. Electron. Commun. (AEÜ)

journal homepage: www.elsevier.com/locate/aeue



Regular paper

Deep reinforcement learning and Bayesian optimization based OpAmp design across the CMOS process space

Eleni Papageorgiou [a,*], Andi Buzo [b], Georg Pelz [b], Thomas Noulis [a,c]

[a] Department of Physics, Aristotle University of Thessaloniki, 54124, Greece
[b] Infineon Technologies AG, Munich, Germany
[c] Center for Interdisciplinary Research and Innovation (CIRI-AUTH), Thessaloniki, Greece

ARTICLE INFO

Keywords:
Analog design
Reinforcement learning
Bayesian optimization
Automation
OpAmp design

ABSTRACT

In this work, we propose a Deep Reinforcement Learning (DRL)-based method for the multi-objective optimization of circuit parameters. The approach leverages a custom Reinforcement Learning environment, enhanced with memoization techniques to minimize simulation iterations and improve efficiency, alongside Bayesian Optimization to narrow the design space. This method generates multiple solutions that not only meet specific performance targets but also surpass them, allowing designers to select the most suitable option based on performance trade-offs. The approach is validated on a two-stage operational amplifier topology, implemented across three different process nodes: 22 nm, 65 nm, and 180 nm. The resulting solutions create a visualization of the design space, offering intuitive and reliable insights into key performance metrics and design trends derived from the agent's exploration. By integrating this DRL-based approach into the analog circuit design workflow, the time-to-market is significantly reduced, while the method enhances the capabilities of design experts by automating parameter selection and optimization.

1. Introduction

The continuous scaling of the integrated circuits (ICs) and the demand for minimizing the time-to-market leads to the need of design cycle acceleration. The complexity of modern semiconductor chips makes the designing unmanageable without the use of Electronic Design Automation (EDA) tools. There are wide-known tools that are used for simulation, design-synthesis and verification regarding digital design [1,2]. The introduction of Hardware Description Languages (HDLs) revolutionized the design process by allowing designers to focus solely on functionality [3]. The ability to utilize abstract representations of standardized electronic components is a crucial factor for maximizing efficiency.

On the other hand, analog circuits have a lot more free parameters than the digital, making them more complex and difficult to apply automated approaches, especially for synthesis. The design cycle includes, consecutively, specifications definition, topology selection, sizing of the circuit, layout, parasitics simulations, prototype fabrication, testing and finally production [4].

Accelerating the design cycle of analog integrated circuits (ICs) remains challenging due to the complexity, the interdependence of circuit parameters and the vast design space. Experts accumulate this expertise over many years, having iteratively refined various circuits across multiple projects. Despite the repetitive nature of the procedure, transferring this knowledge to another engineer is difficult due to the high sensitivity of analog circuits, even for similar topologies. Consequently, designers often have to start from scratch when assigned a new design, rendering the process inefficient and unproductive.

Several automation methodologies have emerged to address this obstacle, broadly categorized into knowledge-based and optimization-based approaches. Knowledge-based methodologies capture the expertise of human designers in algorithms that can be executed by a computer [5–7]. This approach, however, requires human intervention and is often time-consuming, as it involves translating heuristic and intuitive thought processes into explicit algorithms.

Optimization-based methods are divided into model-based and simulation-based approaches. Model-based methods use mathematical expressions to capture the expert knowledge or circuit behavior, enabling the extraction of objective functions for optimization techniques to converge on solutions [8–11]. However, due to the extensive human effort required, the automation potential of these methods is limited.

In contrast, simulation-based methods do not rely on prior knowledge to represent circuit behavior but instead learn it through simulations. The human-like behavior of learning through the results lead the researches to the use of heuristic approaches. Examples include

* Corresponding author.
E-mail address: epapagea@auth.gr (E. Papageorgiou).

https://doi.org/10.1016/j.aeue.2025.155697
Received 26 November 2024; Accepted 23 January 2025
Available online 30 January 2025

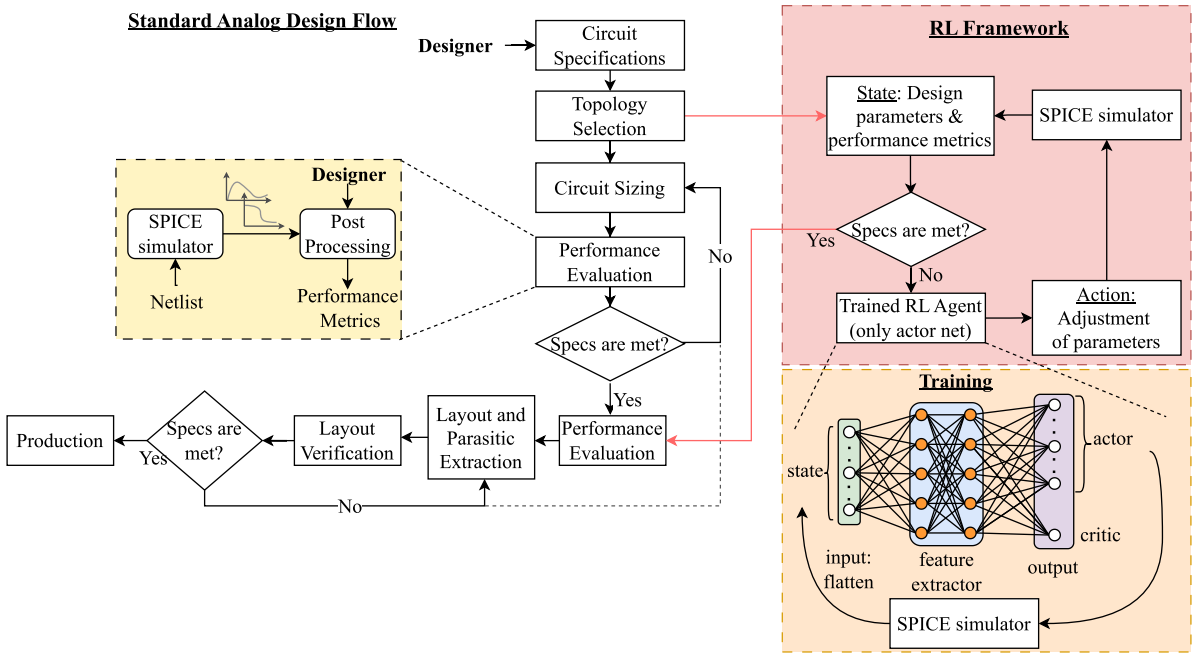


Fig. 1. Analog design standard flow versus proposed RL framework.

Bayesian optimization (BO) [12–14], simulated annealing [15], Evolutionary Algorithms (EAs) [16–18] and Genetic Algorithms (GAs) [19, 20].

In recent years, the rapid increase in computational power has created opportunities for the rise of Machine Learning (ML) methods. ML methods are used in multiple fields, as well as in the EDA flow [21]. In [22], the authors are replacing the SPICE simulator with multiple Neural Networks (NNs) to accelerate the GA algorithm. [23] investigates the performance of Reinforcement Learning (RL) in parameters extraction of a BSIM model. In [24], a graph neural network is utilized to convert circuit schematic into graphs, in order to predict parasitics. The authors of [25] proposed a machine learning optimization framework trained using deep reinforcement learning for circuit parameters selection.

In this article, we present a more detailed analysis of [26], our previous work, which was extended by the use of memoization and BO [27] to leverage efficiency and automation. Memoization is a technique to replace high-cost functions, like simulations, with already gathered data. BO is a sequential method for finding the global maximum with as few iterations as possible, using probabilistic models. It is utilized to scan, with a few simulations, the design space and find a range of potential functional set of parameters. With this narrowed space, the device sizing phase of the design cycle is addressed using an RL framework with a custom environment. In addition, a general-purpose Python API for the circuit simulator is developed, which accepts the design parameters and the netlist as input and returns the performance metrics. The aim is to derive insights into the given topology without relying on human intervention or prior knowledge, offering a comprehensive representation of the design space and potential solutions for designers to choose from based on application-specific requirements and trade-offs.

2. Reinforcement learning framework

The standard analog design flow involves multiple steps and numerous iterations to ensure correct functionality at each phase. As illustrated in Fig. 1, the proposed RL method aims to streamline the circuit sizing process by minimizing the manual effort required from a designer. In the traditional flow [4], tuning circuit parameters is highly time-inefficient, as it necessitates multiple iterations of adjusting parameter sets, followed by simulations that produce non-intuitive results. Designers must then perform extensive post-processing to evaluate multiple diagrams and performance metrics to derive useful insights for further adjustments. This process is not only inefficient, but must be repeated several times throughout the design cycle. Thus, minimizing design cycle time is crucial, and the use of a tool that can efficiently explore the entire design space—in terms of both parameters and performance—is essential to aid designers.

The presented RL framework takes the topology and the design parameters as input. By creating a custom environment, a DRL agent is trained to adjust parameter values. The objective is to minimize the number of steps or simulations required to converge to functional solutions based on the initial specifications. The trained agent gains experience in understanding the design space and efficiently navigating through it. The interaction with the simulator is automated, making the procedure entirely human-free. In contrast with the standard approach, this framework offers multiple solutions with less simulations and without a human-in-the-loop.

2.1. Bayesian optimization

BO is a strategy for efficiently sampling a black-box function with the goal of finding its optimal solution by approximating the function through a probabilistic model, such as a Gaussian Process. In the case of circuit sizing, there is a need for the fast identification of the range of parameters to be explored. Without prior knowledge, this parameter space can be enormous and difficult to manage. Therefore, narrowing the space is essential. The goal is to determine a range of values rather than a specific value, which makes BO particularly well-suited to this application.

The RL agent requires specified value ranges for each adjustable circuit parameter. The BO setup utilizes a Gaussian Process with a Lower Confidence Bound (LCB) acquisition function, which tends to favor exploitation. The objective function is defined to ensure minimum circuit functionality. For instance, for a two-stage OpAmp, the objective function minimizes the number of transistors that are not in saturation. In each call, the process involves modifying the parameters in the netlist, running simulations, and post-processing the simulation results

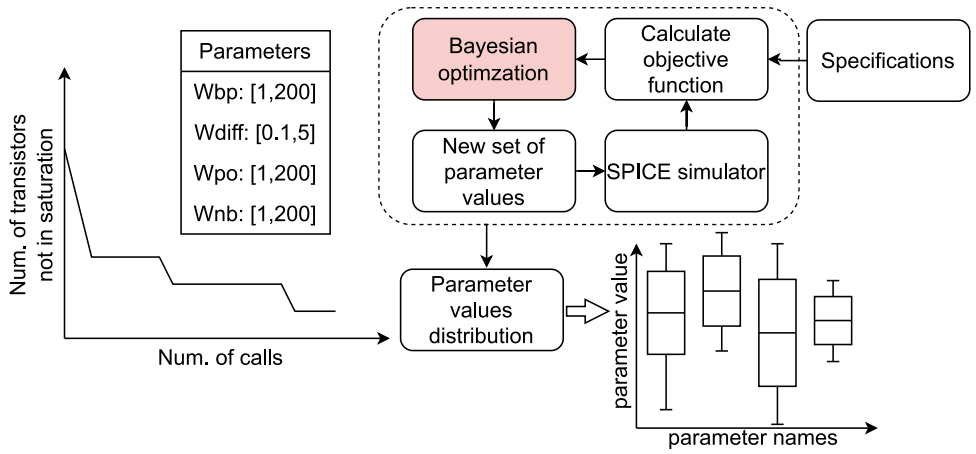


Fig. 2. Analog design standard flow versus proposed RL framework.

to extract the necessary data for the objective function. Fig. 2 depicts how the BO is utilized based on the above description.

From this optimization process, the value distributions of parameters that lead to the minimized objective function are identified. This information is then provided to the RL agent, which narrows the design space, allowing it to fine-tune the parameters within a range known to yield viable solutions. However, as will be discussed further, the design space is not strictly confined to these functional ranges. This flexibility enables the agent to explore further, gaining a deeper understanding of the complex interrelationships between parameters and performance.

2.2. Reinforcement learning

RL is a subcategory of Machine Learning, alongside Supervised and Unsupervised Learning [28]. Unlike the other two subcategories, RL does not require a dataset to be applied. Instead, it follows a trial-and-error approach by interacting with an environment, mapping situations to actions with the objective of maximizing a reward signal [29]. This mapping is called policy, and the goal is to extract the optimal one.

The environment is modeled based on a Markov decision process, which is an analytical framework for decision-making systems [30]. The three fundamental components of this structure are the state, action, and goal. In our setup, the environment is for interacting with a SPICE simulator to understand the design space and train the agent to make the most efficient adjustments to circuit parameters, thereby achieving the desired performance. The custom environment was created using the gymnasium API [31].

2.3. State

The state of the environment is a combination of the performance metrics, parameter values, and the operation region of the transistors. To obtain these observations, a Python interface is created to interact with a SPICE simulator. Given a netlist and a list of parameters, the interface can return the simulator's output, which is then post-processed to extract the relevant performance metrics. The parameter values are included to provide an understanding of the agent's current position within the design space. Additionally, the operation region of the transistors is given to ensure the proper functionality of the circuit based on the application.

2.3.1. Initial state

An initial state must also be defined. The proposed framework is episodic, meaning that there is a defined initial state and a termination state, which marks the end of an episode. At the start of each new episode, the initial state is reset. In this work, the initial state consists of a random set of parameter values and the performance metrics returned after one interaction with the simulator.

2.3.2. Termination state

The termination state, on the other hand, is either achieved by reaching the specified performance metric values or by reaching the maximum number of steps/simulations. Once the thresholds defined for the specifications are met, the agent continues the search until no further improvements are observed. A limit on the possible steps was chosen to emphasize the importance of finding a solution quickly and to avoid investing in a suboptimal policy. The number of steps is also a hyperparameter that can take different values based on the difficulty of the case.

2.4. Action

The action in this framework involves a discrete adjustment of each parameter. For example, for a transistor width, an action might involve increasing or decreasing the width by 0.1 μm. It is also possible to choose not to adjust the value at all, making the possible actions for each parameter three in total, increase, decrease, or stay the same. The number of actions corresponds to the number of parameters, and in each iteration, all parameters are adjusted simultaneously according to the chosen actions.

2.5. Reward

The reward function was shaped [32] to incorporate domain knowledge, while being careful not to make it overly specific, in order to preserve generalization. It incorporates all specifications based on their significance, as well as the operating region of the transistors. Specifications are divided into constraints (C) and objectives (O). The transistor operating regions are of paramount importance, serving as strong constraints to remain in saturation. In Eq. (2), a penalty function is described based on the region of the transistor.

The function is outlined in (1), where β, h_c, h_o are scaling constants for each term, $y, \hat{y}$ is the target and predicted value respectively. The selection of a piece-wise function aims to incentivize the agent to prioritize meeting the most crucial specifications, with a particular emphasis on ensuring transistor regions are addressed first. For the first branch of the function, the difference between the predicted and the target values for the set of the constraints it is only added if the prediction has not yet reached the threshold. Once these are reached among with the regions of the transistors, the reward becomes positive. This transition is governed by a condition ($cond.$) outlined in (3). By shaping the reward function we can guide the agent to aim for reaching the specifications and surpass them.

$$r = \begin{cases} -\beta \sum_{i=1}^{n} \delta_i - h_c \sum_{i \in C} \max(0, y_i - \hat{y}_i) + h_o \sum_{i \in O} o_i, & \text{cond.=F} \\ h_c \sum_{i \in C} |y_i - \hat{y}_i| + h_o \sum_{i \in O} o_i, & \text{cond.=T} \end{cases} \tag{1}$$

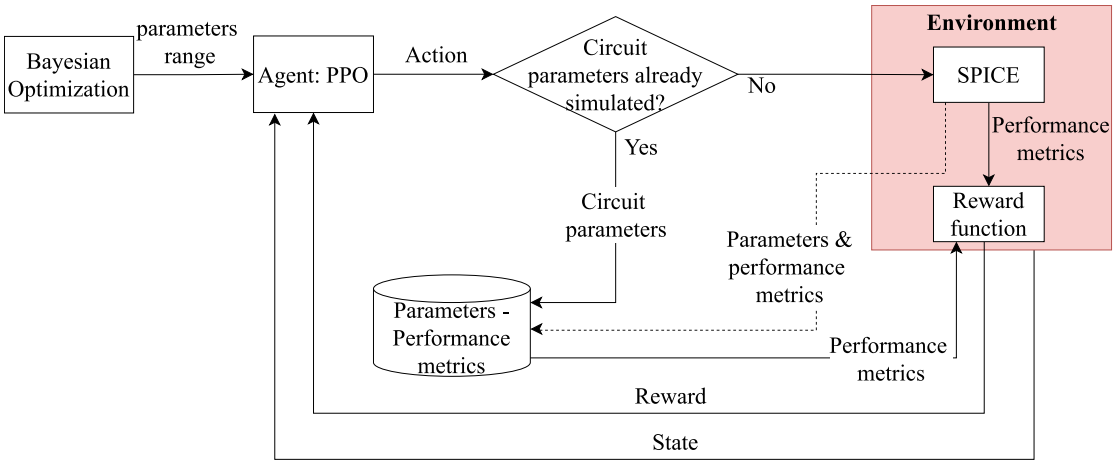


Fig. 3. RL framework.

$$\delta_i = \begin{cases} 0, & \text{operation region of } i\text{th transistor is the desired} \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

$$cond. = True \iff \sum_{i=1}^{n} \delta_i = 0 \,\&\, \hat{y}_i > y_i \forall i \in C \tag{3}$$

2.6. Agent

The agent is the model that receives the state of the environment as input and returns the action as output. For this work, the Proximal Policy Optimization (PPO) algorithm [33] was used to train the model. PPO was chosen because it is straightforward to implement, requires less fine-tuning, and is robust enough to handle a versatile environment that includes both discrete and continuous action spaces.

The algorithm was adopted from stable-baselines3 [34], an open-source framework for implementing Reinforcement Learning algorithms. Also, the hyperparameters of the agent where tuned using BO [35].

2.7. Memoization

The bottleneck of the framework is the SPICE simulation, which requires several seconds to execute, in contrast to the microseconds or milliseconds needed for the calculations of the DRL algorithm. To address this, the memoization technique [36] was utilized. Since the results of the simulation are deterministic, they can be stored and later retrieved to avoid re-running the simulation. Consequently, as the agent explores the design space, a dataset is created that can be used for future training and evaluation. Initially, the dataset is not useful for a newly encountered topology at the start of training. However, as the agent begins to understand the design space, the functional region will be identified. Consequently, as the design parameters adjust to converge toward this region, it becomes likely that the same values will be revisited making the training less time consuming.

A description of how memoization is incorporated to the framework is presented in Fig. 3 and the training is summarized in Algorithm 1 pseudocode.

The framework is designed to be highly flexible and easy to adjust. To apply it to a different topology, only the state needs to be modified, along with categorizing the specifications into objectives and constraints to align with the reward function.

Algorithm 1 Training

```
cur_timestep ← 0
while cur_timestep < max_timesteps do
    parameters ← random()
    results ← run_simulation(parameters)
    metrics ← calculate_metrics(results)
    state ← [metrics, parameters]
    terminated ← False
    step ← 0
    while (step < max_steps) AND (NOT terminated) do
        action ← current_policy()
        parameters ← update_parameters(action)
        if parameters in memoization_file then
            metrics ← memoization(parameters)
        else
            results ← run_simulation(parameters)
            metrics ← calculate_metrics(results)
            memoization_file ← [parameters, metrics]
        end if
        state ← [metrics, parameters]
        reward = reward_function(metrics)
        if reward > 0 then
            if No change in parameters then
                terminated ← True
            else if No improvement then
                terminated ← True
            end if
        end if
        if (cur_timestep % update_agent_steps) then
            policy ← PPO()
        end if
        step ← step + 1
        cur_timestep ← cur_timestep + 1
    end while
end while
```

3. Experiments and results

The functionality of the framework is demonstrated on a two-stage operational amplifier (Fig. 4). The circuit parameters of interest are highlighted, and the remaining parameters are fixed to an initial value. The action and observation spaces are normalized based on the process node. For the specific topology, the constraints and the objectives are given in Eqs. (4) and (5), respectively. In these equations, the gain is

Table 1
Initial constraint thresholds.

Constraint	Threshold
Gain	30 dB
Unity gain bandwidth	1 MHz
Phase margin	45°

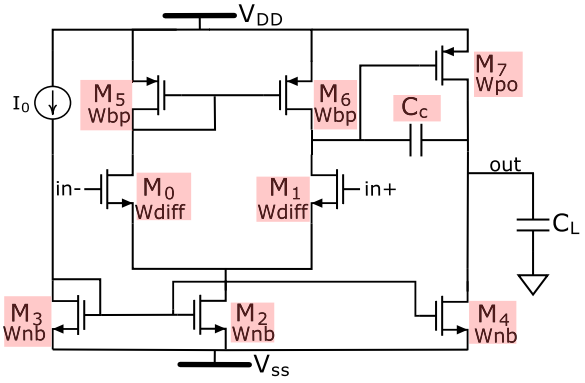


Fig. 4. Two stage operational amplifier [37].

denoted as G, phase margin as PM, unity gain bandwidth as $UGBW$, output voltage swing as $Vout$, input common mode range as $ICMR$ and slew rate as SR. These specifications are selected based on their importance in terms of how they affect the circuit behavior and what a designer is looking to achieve while designing such a topology.

The process was utilized in three different process nodes, 22 nm, 65 nm, and 180 nm. The intention is to evaluate both the functionality of the agent as well as the ability to understand design space. By applying the method on the different process nodes, valuable insights can be gathered.

The reward function hyperparameters were set to $\beta = 5$ and $h_c, h_o = 100, 0.01$. All data was generated using a machine equipped with an Intel Xeon(R) Gold 5218 CPU and 125 GB of memory.

The training was done with the same thresholds for the constraints, presented in Table 1, supposing no prior knowledge of the characteristics. After validating the design space based on the information gained by the agent, harder constraints values are applied for the evaluation, based on the distribution of the performance metrics to extract better performance in each process.

$$C = [G, UGBW, PM] \tag{4}$$

$$O = [\frac{1}{|y_{ICMR} - y_{\hat{ICMR}}|}, \frac{1}{|y_{Vout} - y_{\hat{Vout}}|}, SR] \tag{5}$$

3.1. Training

For the training, the default values for the network architecture are used. The inputs are flattened and then passed to a shared feature extractor, creating a feature vector that is then split into two networks, the actor and the critic. Both the actor and critic networks consist of two dense layers with *tanh* as activation functions. Finally, each the network conclude to a different head. The actor outputs the probabilities for possible actions (increase, decrease, or stay the same) for each of the five parameter leading to 15 outputs and then the action with the highest probability is chosen deterministically. The critic network, on the other hand, outputs the value function that estimates the current state's value.

To train an RL agent for circuit sizing, there is need to specify the parameter ranges. The values of the non-adjustable circuit parameters are depicted in Table 2. The parameter value range for the transistor

Table 2
Combinations of hyperparameters for each process node.

Non-adjustable parameters	22 nm	65 nm	180 nm
V_{dd} (mV)	400	600	900
V_{ss} (mV)	−400	−600	−900
I_o (μA)	15	30	30
C_L (pF)	10	10	10

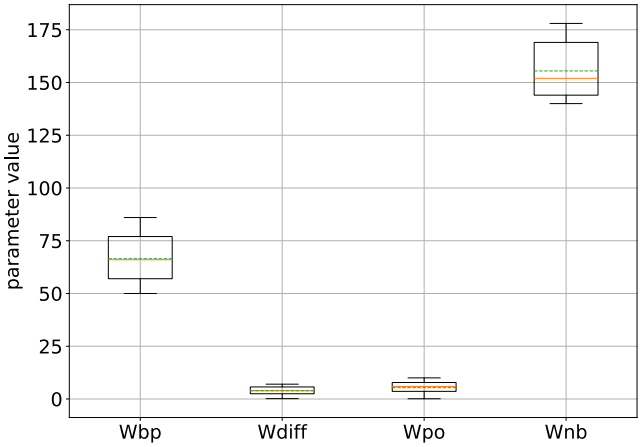


Fig. 5. Distribution of parameter values for 180 nm process.

widths is extracted using BO with Gaussian Process. The C_c is not related to the operation region of the transistors thus was excluded from this phase. The total number of function calls is limited to 100. By reviewing the BO results, the beneficial range of parameter values was identified. The range of each design parameter and its corresponding action step is shown in Table 3.

For clarity, the distribution of the parameters for the 180 nm process is depicted in Fig. 5, as the results for the 22 nm and 65 nm processes are qualitatively equivalent. The orange line is the median and the dotted green line is the mean of the data. Although the W_{diff} parameter was initially set to a range of [1–200], the results consistently showed $W_{diff} = 1$ μm, leading to a smaller range being explored as shown in Fig. 2. After running BO again, the final parameter ranges were used as input for the RL agent. This approach is equivalent to narrowing the design space based on either the designer's knowledge or intuitive equations. Therefore, the proposed solution eliminates human intervention by automatically selecting the parameter ranges based on the given specifications.

BO was also employed to select the hyperparameters of the PPO agent. Each hyperparameter combination was used to train the agent for 10 000 steps for the 22 nm process node. After each training session, the agent was validated over 10 episodes to compute the mean reward and mean episode length. The combinations that achieved the highest mean reward and the shortest mean episode length were selected. The selected hyperparameters and their corresponding performance are presented in Table 4. Detailed information about each hyperparameter can be found in the Stable-baselines3 documentation [34]. The same hyperparameters were applied to the other process nodes.

The maximum number of steps per episode was set to 20 and the training was stopped at the 125,000 total steps if the performance of the agent was not satisfactory.

The training progress for all process nodes are shown in Fig. 6. Fig. 6(a) shows the mean reward calculated in each episode, while Fig. 6(b) shows the length of each episode in terms of steps. The wanted behavior is for the cumulative reward to converge to as high values as possible with positive values, meaning that in most of the simulations that took place, the performance metrics are above the thresholds. For the length of the episodes, the goal is to minimize it achieving as few

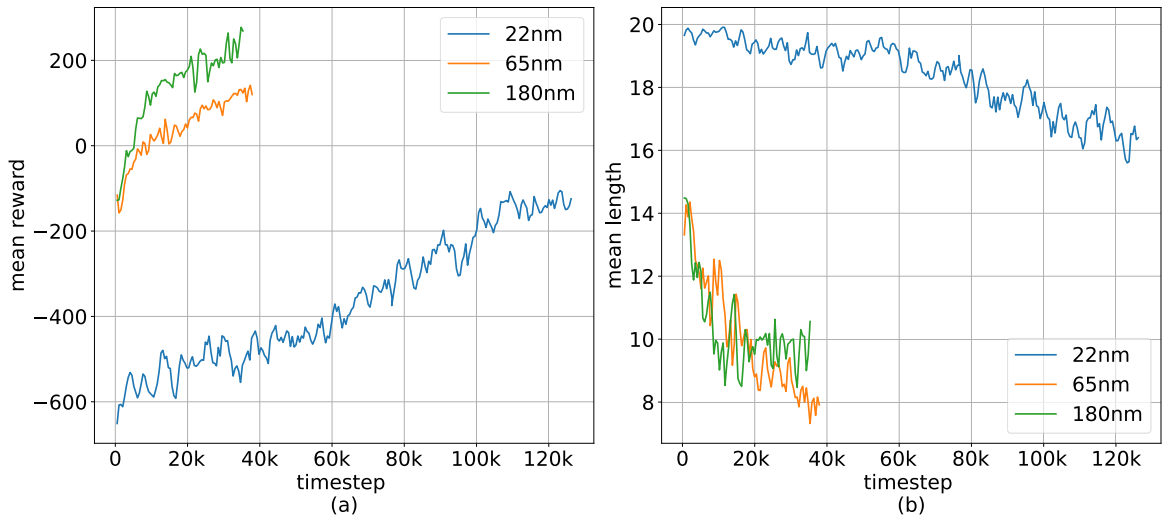


Fig. 6. Training progress (a) mean reward per episode (b) mean episode length per episode.

Table 3
Design parameters configuration for each process node.

	Cc(pF)		Wbp (μm)		Wdiff (μm)		Wpo (μm)		Wnb (μm)		Total num.
	Range	Step	Range	Step	Range	Step	Range	Step	Range	Step	of possible sets
22 nm	[0.1,2]	0.1	[0.1,3]	0.1	[0.1,3]	0.1	[0.1,10]	0.5	[0.1,10]	0.5	7.2e6
65 nm	[1,20]	1	[1,20]	1	[0.3,3]	0.1	[1,20]	1	[10,40]	1	7.2e6
180 nm	[20,40]	1	[55,75]	1	[0.4,6.4]	0.2	[0.5,10.5]	0.5	[145,175]	1	7.2e6

Table 4
Combinations of hyperparameters for each process node.

Hyperparameters.	Values
γ (discount factor)	0.91
λ (GAE)	0.98
ent_coeff	0.09
vf_coeff	0.5
Learning rate	3e−4
n_steps	256
Mean reward	−303
Mean length	16

Table 5
Successful episodes across process nodes.

Process nodes.	22 nm	65 nm	180 nm
Percentage of successful episodes	79	100	100
Average # of steps	14.54	8.66	9.87

simulations as possible, ignoring the use of memoization. Therefore, the fewer steps required to complete an episode, the more effective the process becomes.

3.2. Evaluation

The main objectives during evaluations is to assure the functionality of the agent, the ability to understand the relationships between parameters and performance metrics. For evaluation, 100 episodes are executed. At the start of each episode, the parameters take random initial values. The agent has 20 steps to converge to positive reward values otherwise the episode in truncated and reset back to the initial state. This is done to evaluate the ability of the agent to move in the design space from multiple starting points verifying its functionality.

After evaluating the three agents, each for one process node, a new evaluation without re-training and new constraints configuration took place. The values of the new thresholds are determined by the distribution of the performance metrics gathered from the successful episodes.

3.3. Results discussion

The performance of the agent is examined based on its ability to understand the design space and be able to navigate through it.

In Table 5, the percentage of successful episodes for each process and the average number of steps needed for these episodes are shown. Due to the limited number of steps, the agent may not be able to reach a successful termination state. For the 22 nm process, this is the case for the 11 unsuccessful episodes where the reward tends to be maximized but 20 steps are not enough.

Also, it is noticeable that for the most complex node, 22 nm, the average number of simulations are more. This is expected as a designer would also needed more simulations to accomplish the specification for such process node, due to the way more advanced compact model used and the way more device parameters and related simulated effects. It is totally different when a circuit is simulated with BSIM3 instead of BSIM — SOI, PSP and BSIM5.

An entire episode iteration is shown in Fig. 7, illustrating the behavior of a trained agent on the 65 nm process node. For this instance, the total number of simulations required was 9. By the 5th step, the agent achieved a non-negative reward and continued exploring the design space for further optimization. The episode concluded when the reward stopped increasing, with the last step yielding a slightly lower reward. During this process, both the Unity Gain Bandwidth (UGBW) and Gain (G) continued to increase, while the Phase Margin (PM) decreased but remained above the defined threshold to compensate for the trade-off. On the other hand, most of the objectives tended to settle within a relatively narrow range, except for the Slew Rate (SR), which appeared to increase. The performance thresholds for this instance are stated in Table 6. The first reward is defined arbitrarily to zero as is a step is need to calculate the reward and thus is ignored.

The agent's functionality is verified not only by the reward convergence but also by the diversity of actions taken, indicating a lack of

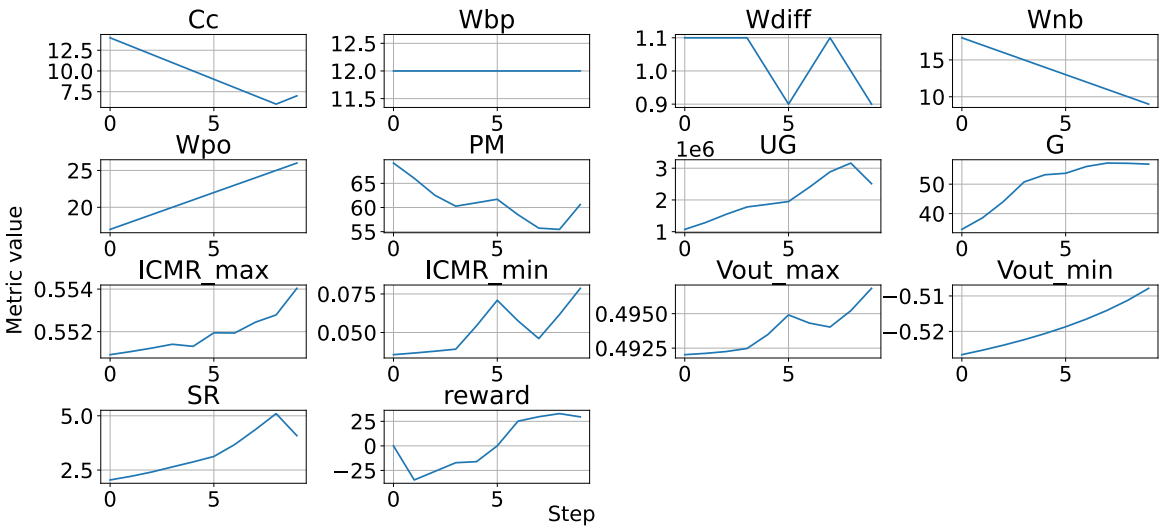


Fig. 7. One completed episode example (65 nm).

Table 6
Constraint thresholds after first evaluation.

Constraint	22 nm	65 nm	180 nm
Gain	30 dB	50 dB	70 dB
Unity gain bandwidth	5 MHz	2 MHz	1 MHz
Phase margin	45°	45°	45°

Table 7
Comparison with state-of-the-art simulation based methods.

Methodology	GA [20]	BO [14]	GP [10]	RL [25]	This work
Execution samples	7550	150	2700	27	10
Training samples	–	–	–	10 000	20 000
Process	180 nm	180 nm	65 nm	45 nm	180 nm

Table 8
Mean values of specification achieved.

Specification	22 nm	65 nm	180 nm
G (dB)	50.96	57.30	80.00
$UGBW$ (MHz)	9.78	4.68	1.56
PM (°)	49.17	46.50	60.70
$ICMR$ (V)	[0.11,0.39]	[0.07,0.593]	[0.005,0.81]
$Vout$ (V)	[−0.25, 0.39]	[−0.44,0.51]	[−0.79,0.84]
SR (V/μs)	7.84	7.50	1.34

bias. A biased behavior would involve sticking to one action, such as continuously decreasing a parameter throughout the episode without regard to the reward progression. However, such a scenario could still occur even if the agent was not biased. Which is why this particular episode was chosen to demonstrate the agent's ability to simultaneously adjust parameters in various ways. In this example, it is clear that multiple actions are selected for each parameter, making the agent's behavior resemble that of a human expert.

After the evaluation of all the process nodes, the statistical analysis of the performance metrics are shown in Fig. 9 with box plots. As expected the gain tends to get higher values in higher process nodes and on the other hand bandwidth tends to get lower values. The agent is able to demonstrate the differences between process nodes and give insights, helping during the migration of topologies. Based on these distributions, the new thresholds are determined Table 6.

With the new constraints, 3D scatter plots are generated, as shown in Fig. 8, where each axis represents one of the three constraints. Each point is colored according to the corresponding reward values for the specific set of parameters. Observing the color gradients makes it easier to identify the most suitable combination of values based on the application requirements. These new constraints were applied to focus on gathering high-performance solutions based on the process characteristics. Notably, the agents were not retrained for these new constraints; instead, they were evaluated as-is. Despite having no prior knowledge of the specific process node, the agent was able to autonomously generate solutions that not only adhered to the constraints but also matched the characteristics and capabilities of each process node.

The comparison in Table 7 shows that while the proposed approach may require more samples for training than [25], which also uses RL, the evaluation of the agents demonstrates that this approach accelerates convergence. For both RL methods, the samples are categorized into training and execution (evaluation) samples, whereas for other methods, only a single category of samples is presented. Additionally, compared to more traditional methods like GAs and [10] that employ geometric programming (GP), the proposed method is superior, despite the time-consuming training phase, as it results in generalized and widely applicable models.

Table 8 shows the mean value for each specification. The means were calculated using the 10 most high rewarding parameter sets. It is possible for the user to tune the hyperparameter of the reward function to achieve different performances.

4. Future work

The proposed method has been demonstrated on an operational amplifier, a linear circuit. However, its application is not limited to this specific topology. By modifying the constraints and objectives, the method can be extended to other circuit types or vehicles. While the training and execution time are expected to vary with the complexity of each circuit, the feasibility of the approach remains intact. Additionally, the framework offers flexibility in narrowing the design space. For instance, in this study, constraints were tailored to focus on specific performance metrics, but the same method could also be adapted to prioritize other design considerations, such as enforcing the operation of transistors in the subthreshold region rather than saturation. Future research on the above topics will further explore the method's potential to better understand its capabilities and extend its applicability.

5. Conclusion

In this paper, we proposed a Deep Reinforcement Learning (DRL) framework for selecting analog circuit parameters. By incorporating

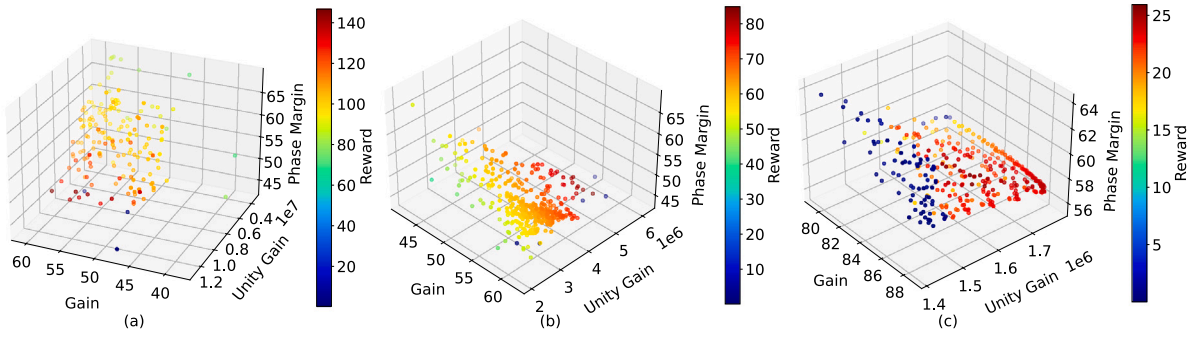


Fig. 8. 3D heated scatter plot for (a) 22 nm, (b) 65 nm and (c) 180 nm process nodes.

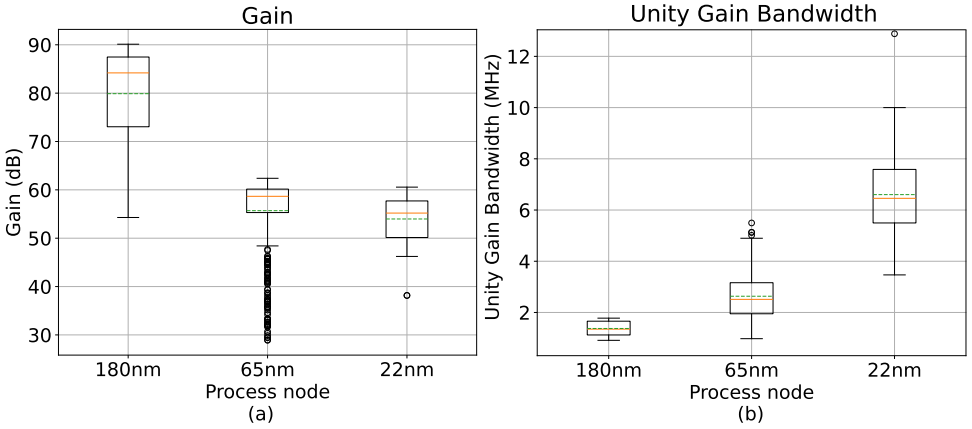


Fig. 9. Box plots for each constraint across all process nodes.

Bayesian Optimization (BO) and memoization techniques, the agent can effectively understand and navigate an unknown design space. This approach not only provides solutions but also offers valuable insights into both the circuit topology and various process nodes. Compared to other approaches, our framework demonstrates improved generalization and faster convergence times, serving as a useful tool for minimizing the design cycle.

CRediT authorship contribution statement

Eleni Papageorgiou: Conceptualization, Software, Writing – original draft. **Andi Buzo:** Writing – review & editing, Supervision, Conceptualization. **Georg Pelz:** Writing – review & editing, Supervision, Conceptualization. **Thomas Noulis:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Co-author, Prof. Thomas Noulis serves as associate editor to the journal. Link : https://www.sciencedirect.com/journal/aeu-international-journal-of-electronics-and-communications/about/editorial-board

If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This scientific paper was created within the framework of "Important Projects of Common European Interest on Microelectronics and Communication Technology" in collaboration between Infineon and the Department of Physics at the Aristotle University of Thessaloniki (AUTh).

Data availability

The data that has been used is confidential.

References

[1] O'Loughlin D, Coffey A, Callaly F, Lyons D, Morgan F. Xilinx vivado high level synthesis: Case studies. 2014.
[2] Synopsys. VCS® functional verification solution. 2024, https://www.synopsys.com/verification/simulation/vcs.html. [Accessed 2 October 2024].
[3] Wakerly JF. Digital design: principles and practices, 4/E. Pearson Education India; 2008.
[4] Baker RJ. CMOS: circuit design, layout, and simulation. John Wiley & Sons; 2019.
[5] Carley LR, Rutenbar RA. How to automate analog IC designs. IEEE Spectr 1988;25(8):26–30.
[6] Sheu B, Lee J, Fung A. Flexible architecture approach to knowledge-based analogue IC design. IEE Proc G 1990;137(4):266–74.
[7] Stefanovic D, Kayal M, Pastre M. PAD: A new interactive knowledge-based analog design approach. Analog Integr Circuits Signal Process 2005;42:291–9.
[8] Singh AK, Ragab K, Lok M, Caramanis C, Orshansky M. Predictable equation-based analog optimization based on explicit capture of modeling error statistics. IEEE Trans Comput-Aided Des Integr Circuits Syst 2012;31(10):1485–98.
[9] Zemliak AM. Analog system design problem formulation by optimum control theory. IEICE Trans Fundam Electron Commun Comput Sci 2001;84(8):2029–41.
[10] Chowdhury SR, Bhardwaj S, Kitchen J. Design automation of CMOS Op-amps using statistical geometric programming. In: 2022 IEEE international symposium on circuits and systems. ISCAS, IEEE; 2022, p. 1575–9.
[11] Ozenli D, Alaybeyoglu E, Kuntman H, Cicekoglu O. MOSFET-only filter design automation based on polynomial regression with exemplary circuits. AEU-Int J Electron Commun 2018;84:342–54.
[12] Lyu W, Xue P, Yang F, Yan C, Hong Z, Zeng X, Zhou D. An efficient bayesian optimization approach for automated optimization of analog circuits. IEEE Trans Circuits Syst I Regul Pap 2017;65(6):1954–67.
[13] Lyu W, Yang F, Yan C, Zhou D, Zeng X. Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In: International conference on machine learning. PMLR; 2018, p. 3306–14.

[14] Zhang S, Yang F, Zhou D, Zeng X. An efficient asynchronous batch bayesian optimization approach for analog circuit synthesis. In: 2020 57th ACM/IEEE design automation conference. DAC, IEEE; 2020, p. 1–6.
[15] Gielen GG, Walscharts HC, Sansen WM. Analog circuit design optimization based on symbolic simulation and simulated annealing. IEEE J Solid-State Circuits 1990;25(3):707–13.
[16] Barari M, Karimi HR, Razaghian F. Analog circuit design optimization based on evolutionary algorithms. Math Probl Eng 2014;2014(1):593684.
[17] Vural RA, Yildirim T. Analog circuit sizing via swarm intelligence. AEU-Int J Electron Commun 2012;66(9):732–40.
[18] Shreeharsha K, Siddharth R, Korde CG, Vasantha M, YB NK. An exponential variation based PSO for analog circuit sizing in constrained environment. AEU-Int J Electron Commun 2024;155531.
[19] Jafari A, Zekri M, Sadri S, Mallahzadeh A. Design of analog integrated circuits by using genetic algorithm. In: 2010 second international conference on computer engineering and applications. Vol. 1, IEEE; 2010, p. 578–81.
[20] Harsha M, Harish B. An integrated maxfit genetic algorithm-SPICE framework for 2-stage op-amp design automation. In: 2018 IEEE computer society annual symposium on VLSI. ISVLSI, IEEE; 2018, p. 170–4.
[21] Huang G, Hu J, He Y, Liu J, Ma M, Shen Z, Wu J, Xu Y, Zhang H, Zhong K, et al. Machine learning for electronic design automation: A survey. ACM Trans Des Autom Electron Syst (TODAES) 2021;26(5):1–46.
[22] Wolfe G, Vemuri R. Extraction and use of neural network models in automated synthesis of operational amplifiers. IEEE Trans Comput-Aided Des Integr Circuits Syst 2003;22(2):198–212.
[23] Papageorgiou E, Alia G, Buzo A, Pelz G, Noulis T. MOSFET model parameter extraction using reinforcement learning. In: 2024 panhellenic conference on electronics & telecommunications. PACET, IEEE; 2024, p. 1–5.
[24] Ren H, Kokai GF, Turner WJ, Ku T-S. ParaGraph: Layout parasitics and device parameter prediction using graph neural networks. In: 2020 57th ACM/IEEE design automation conference. DAC, IEEE; 2020, p. 1–6.
[25] Settaluri K, Haj-Ali A, Huang Q, Hakhamaneshi K, Nikolic B. Autockt: Deep reinforcement learning of analog circuit designs. In: 2020 design, automation & test in europe conference & exhibition. DATE, IEEE; 2020, p. 490–5.
[26] Papageorgiou E, Noulis T, Buzo A, Pelz G. Automated design of two-stage op amp using reinforcement learning. In: 2024 31st IEEE international conference on electronics circuits and systems. ICECS, IEEE; 2024, p. 1–5.
[27] Mockus J. The Bayesian approach to global optimization. In: System modeling and optimization: proceedings of the 10th IFIP conference New York City, USA, August 31–September 4, 1981. Springer; 2005, p. 473–81.
[28] Burkov A. Machine learning engineering, vol. 1, True Positive Incorporated Montreal, QC, Canada; 2020.
[29] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT Press; 2018.
[30] Howard RA. Dynamic programming and Markov processes. John Wiley; 1960.
[31] Towers M, Terry JK, Kwiatkowski A, Balis JU, Cola Gd, Deleu T, Goulão M, Kallinteris A, KG A, Krimmel M, Perez-Vicente R, Pierré A, Schulhoff S, Tai JJ, Shen ATJ, Younis OG. Gymnasium. 2023, http://dx.doi.org/10.5281/zenodo.8127026, URL: https://zenodo.org/record/8127025.
[32] Grzes M. Reward shaping in episodic reinforcement learning. 2017.
[33] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv:1707.06347.
[34] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: Reliable reinforcement learning implementations. J Mach Learn Res 2021;22(268):1–8, URL: http://jmlr.org/papers/v22/20-1364.html.
[35] Garnett R. Bayesian optimization. Cambridge University Press; 2023.
[36] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. MIT Press; 2022, p. 390–3.
[37] Allen PE, Holberg DR. CMOS analog circuit design. Elsevier; 2011.