# Practical Solutions to Accelerating ASIC Design Development Using Machine Learning

*Keertana Settaluri*

Technical Report No. UCB/EECS-2022-27
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-27.html

May 1, 2022

# Practical Solutions to Accelerating ASIC Design Development Using Machine Learning

by

Keertana Settaluri

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Summer 2021

Practical Solutions to Accelerating ASIC Design Development Using Machine Learning

Abstract

Practical Solutions to Accelerating ASIC Design Development Using Machine Learning

by

Keertana Settaluri

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Science

University of California, Berkeley

Professor Borivoje Nikolić, Co-chair

Professor Krste Asanović, Co-chair

Application specific integrated circuits (ASICs) are ubiquitous in modern society, with their high performance, low power, low area characteristics aptly utilized in state-of-the-art electronics and devices. As technology nodes scale, however, it becomes increasingly difficult to bring innovation to circuit systems, as the complexity of design rules and the substantial impact of layout parasitics not only delays time-to-market, but is also more expensive. Emphasis is therefore placed on improving the ASIC design flow, with the primary focus of addressing inefficiencies in this manual and iterative process. The implications of faster ASIC development are far-reaching, allowing for rapid prototyping and better scaling in complex and large System-on-Chip designs, reducing overall cost. Improving the chip design process, however, is challenging, as the iterative circuit, layout, and verification stages heavily involve circuit designers, who often use circuit intuition to dictate crucial decisions. Tools that accelerate the chip design process must therefore consider the unique constraints posed by the ASIC domain in order to be practically utilized by circuit designers.

This thesis presents a method for analyzing the efficacy of automated ASIC design tools, specifically by assessing the accuracy, practicability, automation, interpretability, generalizability and run-time efficiency of the algorithm. This is established by presenting one in-depth case study and two projects where machine learning can be used to address inefficiencies in ASIC design, and include: 1) an analog circuit design framework that uses reinforcement learning (RL) to size parameters for a given circuit topology to meet a target specification, 2) an analog sub-clustering tool that uses graphical convolutional neural networks (GCNNs), and 3) using convolutional neural networks (CNNs) to detect defects in circuit yield. The goal is demonstrate that machine learning techniques can not only be successfully used for these three applications, but can be comprehensively analyzed to obtain practical and feasible solutions in circuit design.

Specifically, the results of the RL analog circuit framework show that this solution achieves state-of-the-art run-time efficiency across six unique circuit topologies of varying complexity while considering layout parasitics. Additional analyses is also conducted to explain the decision-making of the algorithm, establishing that the obtained performances are explainable and analyzable in the context of circuit design. Furthermore, for several non-linear circuits, the algorithm obtains initial designs that are better than that of an expert, providing potential for better intuition into the boundaries of performance for these circuits.

The GCNN framework for analog sub-clustering project demonstrates that high run-time efficiency with over 91% accuracy can be achieved while being fully automated, requiring no input from the designer for classification. In addition, the algorithm successfully scales to a variety of analog circuits, which is crucial in establishing practicality. The yield defect detection framework using CNNs shows that ML can be applied to a post-silicon application, successfully resulting in identification of yield defects in real and noisy scan diagnosis tests while reducing the layout search space significantly.

To Mom and Dad

# Contents

**Contents** ii

**List of Figures** v

**List of Tables** viii

**1 Introduction** 1
1.1 Motivation . . . 1
1.1.1 Automation of ASIC design . . . 2
1.2 Prior work . . . 4
1.3 Thesis scope and outline . . . 5

**2 Reinforcement Learning for Well-Defined Circuit Topologies** 7
2.1 Introduction to analog design and sizing . . . 7
2.2 Prior analog sizing tools . . . 8
2.3 Introduction to machine learning . . . 10
2.4 Reinforcement learning framework . . . 10
2.4.1 RL analog sizing setup . . . 11
2.4.2 Trajectory generation . . . 11
2.4.3 RL agent . . . 12
2.4.4 Training with schematic simulations . . . 12
2.4.5 Deployment . . . 14
2.5 Transimpedance amplifier results . . . 16
2.6 Two-stage amplifier . . . 18
2.6.1 Manual design procedure . . . 18
2.6.2 Operational amplifier reinforcement learning setup . . . 20
2.7 Low-dropout voltage regulator . . . 22
2.7.1 Manual design procedure . . . 22
2.7.2 LDO target metrics and action space . . . 24
2.7.3 Training and deployment results . . . 25
2.7.4 Practicability and interpretability analyses . . . 26
2.8 Two-stage operational transimpedance amplifier with negative $g_m$ load . . . 32

2.8.1 Schematic results . . . . . . 32
2.8.2 Layout simulation results . . . . . . 34
2.9 Analysis and conclusion . . . . . . 36

**3 Reinforcement Learning for Parasitic-Sensitive Circuit Topologies 38**
3.1 Modified deployment algorithm . . . . . . 38
3.2 Two-stage transimpedance amplifier with negative $g_m$ load . . . . . . 40
3.2.1 Design equations and expert methodology . . . . . . 41
3.2.2 RL setup . . . . . . 43
3.2.3 Training results . . . . . . 43
3.3 Folded-cascode amplifier . . . . . . 46
3.3.1 Schematic training setup . . . . . . 47
3.3.2 Design equations and expert methodology . . . . . . 51
3.3.3 Schematic training and deployment results for folded-cascode . . . . . . 53
3.3.4 Reachability analyses in schematic trained agent for folded-cascode . 58
3.3.5 Generalization to layout parasitics . . . . . . 63
3.4 Open-source code and framework . . . . . . 71
3.4.1 Setup . . . . . . 73
3.4.2 Code flow . . . . . . 73
3.4.3 Training . . . . . . 74
3.5 AutoCkt analyses and conclusion . . . . . . 75

**4 Reinforcement Learning for Nonlinear Circuit Topologies 77**
4.1 Introduction . . . . . . 77
4.2 Strong-arm comparator . . . . . . 78
4.2.1 Comparator basics . . . . . . 78
4.2.2 Manual comparator design . . . . . . 78
4.2.3 Comparator RL setup . . . . . . 79
4.2.4 Results on training and deploying with schematic simulations . . . . 80
4.3 Ring amplifier . . . . . . 81
4.3.1 RAMP basics . . . . . . 82
4.3.2 RAMP Autockt setup . . . . . . 85
4.3.3 Results on training and deploying with schematic simulations . . . . 86
4.3.4 Results on deploying with layout simulations . . . . . . 88
4.4 Conclusion . . . . . . 93

**5 Analog Sub-Circuit Clustering with Graph Convolutional Neural Networks 96**
5.1 Analog synthesis and sub-circuit extraction background . . . . . . 96
5.2 GCNN framework . . . . . . 97
5.2.1 Feature extraction . . . . . . 98
5.2.2 Cluster count and label extraction . . . . . . 98
5.2.3 Graph creation . . . . . . 98

5.2.4 Graph labelling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
5.3 Experimental setup and results . . . . . . . . . . . . . . . . . . . . . . . 99
5.4 Analysis and conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

**6 Yield Defect Detection using Convolutional Neural Networks 102**
6.1 Defect-prone layout pattern detection . . . . . . . . . . . . . . . . . . . . 102
6.1.1 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 103
6.2 Layout feature learning with convolutional neural networks . . . . . . . . . . 104
6.3 Layout pattern extraction and ranking . . . . . . . . . . . . . . . . . . . . 106
6.3.1 Saliency extraction from learned CNN . . . . . . . . . . . . . . . . 106
6.3.2 Common defect pattern extraction from saliency map . . . . . . . . . 107
6.4 Experimental results with synthetic and real diagnostic data . . . . . . . . . 109
6.4.1 Synthetic diagnostic data results . . . . . . . . . . . . . . . . . . . 109
6.4.2 Real diagnostic data results . . . . . . . . . . . . . . . . . . . . . 111
6.5 Analysis and conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 112

**7 Conclusion 115**
7.1 Thesis contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
7.2 Future directions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116

**Bibliography 117**

# List of Figures

1.1 Productivity and quality gap during ASIC development [3]. . . . . . . . . . . . . 2
1.2 Requirements for automation in ASIC design. . . . . . . . . . . . . . . . . . . . . . 3

2.1 Reinforcement learning algorithm showing that the input to the network are the current performance, target specification, and current parameters, and the output is a probabilistic distribution that is sampled to determine whether to increment, decrement, or retain each circuit parameter. . . . . . . . . . . . . . . . . . . . . 11
2.2 Top level system diagram showing that AutoCkt takes as input a circuit netlist, its simulation testbench, and a target design specification to find parameters for the circuit such that it meets the specification. . . . . . . . . . . . . . . . . . . . 12
2.3 Proximal policy optimization framework setup using Ray [52]. . . . . . . . . . . 14
2.4 Diagram of generator-based design using BAG [55]. . . . . . . . . . . . . . . . . 15
2.5 Simple transimpedance amplifier schematic. . . . . . . . . . . . . . . . . . . . . . 17
2.6 Mean episode reward for transimpedance amplifier. . . . . . . . . . . . . . . . . 17
2.7 Two-stage operational amplifier schematic. . . . . . . . . . . . . . . . . . . . . . 18
2.8 Mean episode reward for two-stage amplifier. . . . . . . . . . . . . . . . . . . . . 20
2.9 Distribution of obtained or learned, reached, and unreached target design specifications. Bottom left shows the 3D plot with three of the four design metrics. The rest of the plots show differing metric combinations to visually demonstrate which design points are not met. . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.10 LDO schematic. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2.11 Mean episode reward for LDO. . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2.12 Met and unmet target specifications for LDO. . . . . . . . . . . . . . . . . . . . 27
2.13 Obtained performances by agent versus target specifications. . . . . . . . . . . 28
2.14 Obtained parameters by agent for reached performances. . . . . . . . . . . . . . 28
2.15 Agent-obtained headroom values for several devices. . . . . . . . . . . . . . . . 29
2.16 Comparing agent versus expert parameter values. . . . . . . . . . . . . . . . . . 30
2.17 Headroom values for expert and agent designs. . . . . . . . . . . . . . . . . . . 31
2.18 Schematic and action space for two-stage op amp with negative $g_m$ load. . . . . 32
2.19 Mean episode reward over environment step for negative $g_m$ operational amplifier. 33
2.20 Distribution of reached target design specifications for the operational amplifier with negative $g_m$ load. Note that this example does not contain any unreached objectives. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

2.21 Top left, top right, and bottom left figures show a sample trajectory for the transferred agent attempting to reach one target design specification. Bottom right shows a histogram plotting difference between schematic and layout simulation. 35

3.1 Parasitic deployment algorithm that shows how schematic and layout simulation distributions are combined to obtain a target specification. . . . . . . . . . . . 39
3.2 Two-stage transimpedance amplifier schematic. . . . . . . . . . . . . . . . . . . . 41
3.3 Mean episode reward for full two-stage transimpedance amplifier. . . . . . . . . 44
3.4 Pipeline ADC block diagram with amplifier circuit in gray. . . . . . . . . . . . 46
3.5 Folded-cascode core topology: 22 parameters total. . . . . . . . . . . . . . . . . 47
3.6 Folded-cascode biasing topology: 20 parameters total. . . . . . . . . . . . . . . 48
3.7 Mean reward per trajectory for 50, 100, and 250 training target specifications. . 55
3.8 Deploying trained agent on target specifications not seen before: folded-cascode. 56
3.9 Mean reward per trajectory for varying number of constraints: folded-cascode. . 57
3.10 Parameter variation for reached performances: folded-cascode. . . . . . . . . . 59
3.11 Overdrive voltage ($V_{ov}$) analysis for obtained performances: folded-cascode. . . . 60
3.12 Metric failure percentage and percent difference to target for those failed metrics, where *f_t* is unity-gain frequency, *A_vo* is gain, *t_set* is settling time, *pm* is phase margin, *P* is power, *V_pp* is voltage swing, and *V_ns* is noise. . . . . . . . . . 61
3.13 Distribution of unreached targets binned by number of metrics that are within their toughest bound. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
3.14 Heatmap of number of occurrences of different metric tradeoffs in unreached target specifications, where *f_t* is unity-gain frequency, *A_vo* is gain, *t_set* is settling time, *pm* is phase margin, *P* is power, *V_pp* is voltage swing, and *V_ns* is noise. 63
3.15 Scatterplot showing the difference between schematic and layout simulation results: folded-cascode circuit. . . . . . . . . . . . . . . . . . . . . . . . . . . 65
3.16 Sensitivity heatmap that shows the absolute percent change in layout metric performance when a selected circuit parameters are individually modified in the design: folded-cascode. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
3.17 Trajectory results when agent is run with a combination of schematic and layout simulations, versus only layout simulations. . . . . . . . . . . . . . . . . . . 67
3.18 BAG-generated LVS-passed layout for the core and biasing folded-cascode circuit for an obtained performance. . . . . . . . . . . . . . . . . . . . . . . . . . 68
3.19 Flow-chart for AutoCkt framework on Github. . . . . . . . . . . . . . . . . . . 74

4.1 Strong-arm comparator schematic. . . . . . . . . . . . . . . . . . . . . . . . . 79
4.2 Mean episode reward for strong-arm comparator. . . . . . . . . . . . . . . . . . 81
4.3 Ring amplifier topology proposed in [71] . . . . . . . . . . . . . . . . . . . . . 83
4.4 Mean reward per trajectory for the ring amplifier. . . . . . . . . . . . . . . . . 86
4.5 Output current versus input voltage schematic simulation plot for the ring amplifier for different designs. . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

4.6 Schematic simulation results for reset and output voltage versus time for the ring amplifier for different designs. . . . . . . . . . . . . . . . . . . . . . . . . . . 88
4.7 Obtained schematic performance plots of output versus input voltage (top), deadzone versus input voltage (middle), and gain versus output voltage (bottom) for the ring amplifier for different designs. . . . . . . . . . . . . . . . . . . . . . . 89
4.8 Scatterplot showing the difference between schematic and PEX simulations for each target metric for the ring amplifier. . . . . . . . . . . . . . . . . . . . . . 90
4.9 Output current versus input voltage layout simulation plot for the ring amplifier for different designs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
4.10 Ring amplifier layout simulation results for reset and output voltage versus time for different designs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
4.11 Obtained layout performance plots of output versus input voltage (top), deadzone versus input voltage (middle), and gain versus output voltage (bottom) for the ring amplifier for different designs. . . . . . . . . . . . . . . . . . . . . . . 93
4.12 Pareto curves for reached performances by agent, along with the expert design. . 94

5.1 Total system overview of the analog clustering algorithm. . . . . . . . . . . . . 97
5.2 GCNN architecture showing two 50 neuron hidden layers, one ReLu non-linearity, and one pooling layer. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99
5.3 Folded cascode with biasing circuit, with hand-labelled clusters. . . . . . . . . . 100
5.4 Output of GCNN algorithm visualized, the black boxes show incorrectly labelled devices, and the similarly-colored circles correspond to the same cluster. . . . . 100

6.1 CNN architecture to learn defect-prone layout feature. . . . . . . . . . . . . . . 105
6.2 Extracting defect patterns (shown in black) with windowing pattern extraction. 107
6.3 System level overview of defect-prone layout pattern-detection algorithm. . . . . 107
6.4 Cell layout (left) and its saliency image (right), with synthetic defect circled. . . 110
6.5 Results of applying windowing technique to saliency results. . . . . . . . . . . 110

# List of Tables

2.1 Comparison of prior analog sizing methods with required simulation count to converge and number of specifications reached versus tested: transimpedance amplifier. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
2.2 Agent-obtained performances and an expert design for the two-stage amplifier with load capacitance of $10pF$. . . . . . . . . . . . . . . . . . . . . . . . . . 23
2.3 RSC and number of specifications reached versus tested: two-stage op amp. . . . 24
2.4 Training and deployment number of specifications reached versus tested: LDO. . 26
2.5 Obtained performances by agent for LDO. . . . . . . . . . . . . . . . . . . . . 27
2.6 Agent versus expert: LDO. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.7 LDO comparison table. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
2.8 RSC and target specifications reached versus tested: two-stage operational amplifier with negative $g_m$ load. . . . . . . . . . . . . . . . . . . . . . . . . . . 33
2.9 RSC and target specifications reached versus tested: two-stage operational amplifier with negative $g_m$ load with layout parasitics. . . . . . . . . . . . . . . . 35
2.10 Summary of circuit topology results presented in this chapter. . . . . . . . . . 37

3.1 Performance range sampled during training for full operational transimpedance amplifier. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
3.2 Obtained schematic performances of trained agent for full operational transimpedance amplifier. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
3.3 Layout performance obtained by agent compared to expert for one design. . . . 45
3.4 Rounded average required simulation count (RSC) for transimpedance two-stage amplifier in schematic and layout for 60 performances. . . . . . . . . . . . . . . 45
3.5 Derived target specifications for folded-cascode in pipeline ADC application. . . 50
3.6 Parameter constraint selection. . . . . . . . . . . . . . . . . . . . . . . . . . . 51
3.7 Comparison of efficiency between agent-obtained and expert-designed performances. 54
3.8 Comparison of prior analog sizing methods with required simulation count (RSC) to converge and number of specifications reached versus tested. . . . . . . . . . 58
3.9 Reached performance ranges by agent ($100o^*$). . . . . . . . . . . . . . . . . . . 59
3.10 Reached performance metric distribution for unreached targets. . . . . . . . . . 60
3.11 Reached performance ranges with layout parasitics. . . . . . . . . . . . . . . . . 67
3.12 Prior work and AutoCkt layout performances for 8-bit and 10-bit pipeline ADC specifications with load capacitance varying from $8 - 30fF$. . . . . . . . . . . 68

3.13 Required Simulation Count (RSC) for AutoCkt with varying seeds (average min/mean/max) and two prior works . . . 68
3.14 FF, SS, SF, SS corner simulation results showing minimum, mean, and maximum obtained metric performances across 60 agent-obtained designs. . . . 70
3.15 8-bit corner simulation results. . . . 71
3.16 10-bit corner simulation results. . . . 71
3.17 Comparison table for prior analog automation works. . . . 72
3.18 Comparison table for folded-cascode topologies. . . . 73

4.1 Performance range sampled during training for strong-arm comparator. . . . 80
4.2 Performance range obtained by agent during training. . . . 82
4.3 Comparison for expert-designed and agent-obtained performances for the same target specification. . . . 82
4.4 Performance range sampled during training for ring amplifier. . . . 85
4.5 Required simulation count (RSC) to converge and number of specifications reached versus tested: ring amplifier. . . . 86
4.6 Obtained schematic simulation performance ranges by agent. . . . 87
4.7 Obtained layout simulation performance ranges by agent. . . . 91
4.8 Expert versus agent-obtained design for same target specification. . . . 92
4.9 Comparison of agent-obtained ring amplifier performance to an expert for the same target specification. . . . 93

5.1 Comparison table showing maximum number of devices in a tested topology, accuracy of algorithm, and automation. . . . 99
5.2 Maximum accuracy of algorithm for multiple circuits. . . . 99

6.1 Accuracy, loss and divergence for the trained CNN. . . . 109
6.2 Test results on simulated diagnosis for different parameters. . . . 111
6.3 Test results on real diagnosis data. . . . 112
6.4 Comparison table for prior yield defect detection works. . . . 114

## Acknowledgments

This dissertation, and my PhD, would not have been possible without the help of so many people. First and foremost, I want to thank my advisors Bora Nikolić and Krste Asanović. Bora, you not only provided me with practical guidance and freedom in pursuing ASIC design automation, but also helped me sell a research vision that was not readily accepted in the circuits community. Despite our differences, you came on board and worked your magic - and my dissertation is better because of it. Krste, you were always available when I needed, and understood my research in the short amount of time I used to describe it to you. You gave valuable input during my qualifying exam, that helped me describe my algorithm better in this dissertation.

I also want to thank my outside committee member, Michael Ranney. The field of education has always fascinated me, and I remember becoming semi-obsessed with education history in your class. Even though I was an outsider with no prior knowledge about the area, you were welcoming and friendly, and seemed to truly care about your students. I appreciate the time you spent (voluntarily) reading my dissertation and meeting with me, the curiosity you had about it and my research, and the questions you asked about my job and general well-being, which was very new to me.

To my mentors Rajeev Jain and Elias Fallon. You both came into my graduate career at really critical points, and your enthusiastic support, positive reinforcement, and selflessness in mentoring a graduate student even before and after I finished my internships at Qualcomm and Cadence, with absolutely no personal gain, really speaks to your characters. My research would not have been the same without your help and support.

I would also like to thank my research collaborators, especially Zhaokai Liu, for providing the necessary (and very well-designed) BAG generators that were crucial to my research. Thanks as well to Ameer Haj-Ali, Jenny Huang, Kourosh Hakhamaneshi, Rishubh Khurana, Arash Mirhaj, and Onur Atan, for coding, mentoring, advising, or all three. Thanks to the other students in Bora's group for providing feedback and friendship.

When I first joined graduate school, I happened upon a tight-knit group of friends in BWRC that have been instrumental during my PhD. Specifically, Antonio, you were a senior-ish student when Emily and I joined, and somehow (maybe it was your constant, very loud and prominent cursing?), we became very good friends. You gave me late-night, post-rager advice on graduate school, while making BWRC feel like home. Ozzy, you were there for those ragers, but I also enjoyed the wonderfully weird conversations we've shared, along with watching basketball games that I didn't really understand.

Emily, we've always joked about where we would be if we didn't meet each other. We've been teammates, desk buddies, roommates, bakers, sometimes bad bakers (though that was mostly me), and shared a lot of incredible memories together. You have a way of perceiving and understanding material and making it intuitive that I've never seen before. They make you so unique, and I'm glad I got to have some of that perspective on my research.

Sidney, Matt, George, Alvaro, Candy, Yessica, Eisha and Anusha: thank you for your friendship and general existence. To my friends from home, who've known me for far too many years: Ben, Misha, and Sasha, thanks for sticking around and providing endless laughs and entertainment.

One of the biggest surprises for me in graduate school was happening upon you, Andy. It's crazy to think, six years ago, we met at visit days not knowing what we would be. Since then, we've shared so many experiences, yet it's still so weird to me that I can enjoy and love even the most mundane things we do together, like going to a store. You've helped me become the strong, independent woman I am today by believing in my abilities, advocating for me when I needed it, and helping me laugh at some of the more unfortunate and awkward events that have happened. I am really excited for the future, but even more so because I know you'll be there with me.

Finally, to my family. Dad, thank you for always pushing me to be ambitious and promoting equality even when circumstances wouldn't allow for it. You introduced me to Electrical Engineering, and gave me so much valuable advice from your time as a PhD student. I truly couldn't have done this without you. Mom, you always care for other people so selflessly. Every day I think about how strong and brave you are, for doing things I couldn't have done myself - from dealing with the loss of loved ones, to risking your life to serve as a nurse, all during the pandemic with a huge smile and positive attitude. I still have a lot to learn from you. And finally, my brother. We somehow ended up in the same lab, doing very similar things - and it's half coincidence and half me being a copycat. I've always looked up to you, not only for your intellect, but having the unique ability to cohesively and correctly express your thoughts, be it technical or anything else. I think you are an amazing leader, and I'm definitely trying to copycat that too.

# Chapter 1

# Introduction

## 1.1 Motivation

Application-specific integrated circuits (ASICs) are ubiquitous in modern society, with their high performance, low power, low area characteristics aptly utilized in state-of-art electronics and devices. As a result, the ASIC market is growing by an estimated 8.6% every year, and predicted to be valued at over $28 billion by 2026 [1]. This higher demand is not only fueling technological innovation, but also reliance on ASICs to provide newer process technologies at smaller form factors, faster speeds and better energy efficiency than their predecessors.

As technology nodes scale, however, it becomes increasingly difficult to bring innovation to circuits, as the complexity of design rules and prominence of layout parasitics not only delays time-to-market, but is also more expensive. In fact, the typical non-recurring engineering (NRE) cost ranges in the tens of millions of dollars, with the development process, consisting of creating goals and constraints, design space exploration, modelling and simulation of the resulting design, creating the physical layout that passes various manufacturing checks, verifying that the circuit functions with this physical design, manufacturing the design, and finally, verifying the design in physical silicon, taking many months to years to complete [2]. In addition, productivity and quality standards are often difficult to meet within a certain time-frame because the development process is iterative, with designers cycling through the different stages many times in order to obtain a feasible design that meets performance specifications. This is illustrated in part in Figure 1.1, where costly chip re-spins alone, which require manufacturing the chip multiple times, significantly delay the development process.

Significant emphasis is therefore placed on improving the ASIC design flow, with the primary focus of addressing inefficiencies in this iterative and manual process. The implications of faster ASIC development are far-reaching, allowing for rapid prototyping and better scaling in complex and large System-on-Chip (SoC) designs, reducing overall cost immensely. Improving the chip design process, however, is challenging, as the iterative circuit, layout, and verification stages heavily involve circuit designers, who often use circuit intuition, or their prior experience to dictate crucial decisions made to the current design. Tools that accelerate the chip design process must consider the unique constraints posed by the ASIC domain in order to be practically utilized by circuit designers.

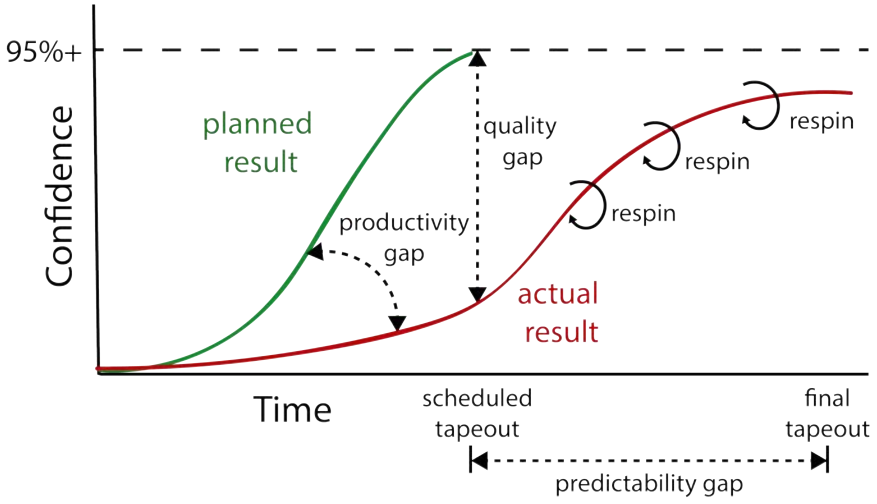


Figure 1.1: Productivity and quality gap during ASIC development [3].

### 1.1.1 Automation of ASIC design

The goal is to automate areas of ASIC development that are innately inefficient or manual, allowing designers to focus on other crucial aspects of the design flow, improving overall development time and cost. Before determining specific algorithms that can be used to address these inefficiencies, it is important to qualify the requirements for an automated tool, that are unique to the ASIC design domain. Figure 1.2 summarizes the six key features, explained below:

- Accurate: the proposed tool must be accurate, meaning that it achieves its objective correctly. The specific ASIC subproblem generally dictates the tolerance for this constraint. For example, automated place and route algorithms often provide initially infeasible designs, even with established CAD tools, versus DRC (design rule check) or LVS (or layout versus schematic) checks must be fully accurate with no errors. The accuracy metric can be represented in many ways, but should be specific to the ASIC problem being solved as opposed to the proposed solution. For example, for a machine learning (ML) algorithm, training or deployment accuracy is traditionally used to assess efficacy [4]. If, however, the proposed ASIC problem is optimization-related, it is also important to also assess the ML solution under this context as well, perhaps via an accuracy metric that compares the ML solution to an expert, or the percent difference between the ML-obtained answer versus a CAD tool.

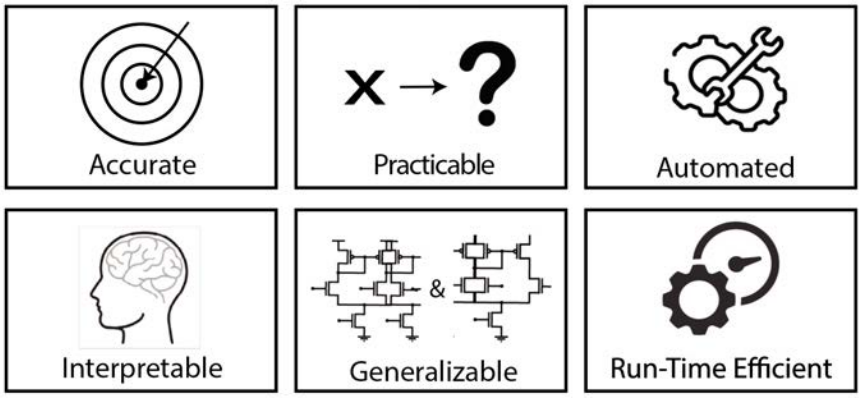


Figure 1.2: Requirements for automation in ASIC design.

- Practicable: the proposed solution must be useful, meaning that within the confines of the ASIC design sub-problem, it should consider all relevant requirements. This includes quantifying any failures and understanding the design space or region where the tool is successful. This analyses not only reveals the limitations of the tool, but can be used to understand its behavior.
- Automated: the tool accelerates the design process compared to traditional methods. The degree of automation can vary considerably, from requiring designers to be heavily involved in the creation and maintenance of the tool's functionality to being fully autonomous with no intervention or hand-holding from the designer.
- Interpretable: the tool should make, to some degree, reasonable assumptions and decisions on the ASIC sub-problem so that it reaches the objective in an interpretable way. This is important, not only as a method to verify the efficacy of the tool, but also to validate and obtain confidence that the tool can achieve objectives reliably. This metric can be derived by comparing designer or CAD tool decision-making and performance to that of the algorithm.
- Generalizable: the tool should function across process technologies, topologies, and SoC designs in a well-defined and scalable manner, ideally with little to no modification of the algorithm. This ensures that the algorithm remains applicable in the changing landscape of ASIC chip design.
- Run-time efficient: the tool should reach the objective in a reasonable amount of time, with nominal compute resources. The constraints vary widely based on the specific ASIC development problem, but in general should require less resources and time than the traditional designer/CAD tool approach. This metric is generally represented in time and required compute machines or operations.

Knowing these six baseline features, and to some degree understanding the trade-offs between them, can not only help in selecting algorithms to address inefficiencies in ASIC design, but also comprehensively assess their efficacy and practicality. The projects covered in this thesis, and their design decisions, are qualified under these metrics.

## 1.2 Prior work

Electronic design automation (EDA) tools are a class of specialized programs solely developed and primarily used to aid in the design of ASICs. Traditional EDA tool-chains [5] encompass multiple facets of the design process, from design methodology and data modeling to physical hardware implementation, manufacturing checks, and logical synthesis. Though these commercially licensed tools are used to partially automate the ASIC flow, they require designers to be heavily involved in process-technology dependent processes, and face long end-to-end run-times due to complex optimization algorithms [6]. Thus, recent research efforts have focused on automating aspects of this existing flow, by either replacing the EDA framework in its entirety, or appending additional algorithms that rely and interface with existing EDA tools. In this section, seminal papers are presented based on frequently occurring themes in the ASIC development process: decision-making, optimization, performance prediction and automation of the end-to-end design flow [7]. These papers are discussed in the context of the six ASIC automation requirements.

Traditionally, decision-making in the ASIC flow is left to the designer, who is responsible for selecting specific tool-chains, hyperparameters and higher-level system design requirements that interface with the EDA framework. Several works address aspects of this decision process, for example using a heterogeneous or deep neural network framework to automate logic synthesis [8, 9], applying Bayesian optimization to select hyperparameters for neural network accelerators [10], or selecting between differing circuit topologies with integer programming [11] or simulation-based approaches [12]. These works enhance automation for their specific sub-problems while demonstrating high fidelity and practicality, and could be explored further to ensure interpretability and generalizability to different technology nodes and unique designs.

Optimization problems in the ASIC design process include applications like deep neural network compression [13], multi-objective tasks such as creating designs to minimize power or other specifications, selecting or generating topologies, or determining parameters for a circuit topology that meet a given target performance. Often, solutions for these design issues are either formulated as numerical or black-box optimization problems. Geometric programming [14] and constraint-based methods have been used in numerical formulations for parameters in a circuit topology and multi-objective design tasks [15], but have not yet been robustly tested for scalability or generalizability, as they are generally applicable in certain specific, and well-defined cases. Evolutionary methods like genetic algorithms and machine learning techniques have been used extensively in designing circuits [16, 17, 18, 19], and are automated, accurate solutions that require more analysis to prove accuracy, reliability, and interpretability.

Performance prediction algorithms accelerate the design flow by providing alternative approximate modeling that provides quicker end-to-end run-time in iterative processes. Examples include power prediction modeling based on a library of pre-modeled components [20], routing congestion prediction with deep graph neural networks [21], clock-tree prognostication using generative adversarial networks [22], or layout-parasitic prediction using machine learning techniques [23]. In general, prediction algorithms range in automation, with the least automated, most manual methods requiring designers to hand-encode accurate models, and the most automated methods demonstrating feasible results but are not extensively tested for interpretability and generalizability.

Full end-to-end design flow automation encompasses the entire EDA flow, and can range from full custom design of digital or analog circuit blocks to larger heterogeneous system-level designs. Generator-based methodologies provide a feasible semi-automated and accurate solution by encouraging designers to manually codify designer intent for a given circuit topology, which can then be reused or regenerated for varying specifications, becoming a more agile process [24, 25]. More ambitious, and still-in-progress initiatives are looking into fully automated end-to-end design using machine learning or other automation techniques [3, 26], but still require accuracy and practicality verification.

In summary, prior automation works provide promising and successful approaches to different aspects of the ASIC design process, and vary vastly in methodology depending on the particular sub-problem they solve. These approaches range in degree of automation, generalizability, accuracy, and interpretability.

## 1.3 Thesis scope and outline

This thesis will focus on how applying various machine learning (ML) techniques can successfully automate different areas of ASIC development, while showing analysis to support that ML can address the six main baseline requirements, which are accuracy, practicability, automation, interpretability, generalizability and run-time efficiency. The goal is to demonstrate that ML techniques have the potential to practically accelerate inefficiencies in the ASIC flow, by presenting one in-depth case study and two initial projects which include: 1) an analog circuit design framework that uses reinforcement learning (RL) to size parameters for a given circuit topology to meet a target specification, 2) using convolutional neural networks (CNNs) to detect defects in physical silicon that affect circuit yield, and 3) an analog sub-clustering tool that uses graph CNNs to group analog circuits. These projects are discussed in the context of the six requirements, and new analyses, metrics, and comparisons are used to validate their efficacy and feasibility. Specifically:

- Chapter 2 introduces and discusses using RL to size a two-stage amplifier and two-stage amplifier with negative $g_m$ load, and demonstrates that this automated framework has high accuracy, is interpretable and run-time efficient.

- Chapter 3 expands these results by testing on a complex parasitic-sensitive two-stage folded-cascode circuit to show that the algorithm is scalable, practicable, and generalizable.
- Chapter 4 discusses results on difficult-to-design ring amplifier and comparator topologies, and further analyzes and validates that the framework is generalizable and accurate.
- Chapter 5 discusses initial results on applying GCNNs to cluster analog sub-circuits to identify critical structures to automate the generation of analog layouts.
- Chapter 6 presents initial results and analyses on applying CNNs to detect systematic layout defects in standard cells for circuit yield.
- Chapter 7 concludes the thesis with overall discussion on the three projects, in addition to providing directions to future work in this area.

# Chapter 2

# Reinforcement Learning for Well-Defined Circuit Topologies

This chapter presents a machine-learning optimization framework trained by using deep reinforcement learning to find post-layout circuit parameters to meet a given target specification, while also gaining knowledge about the entire design space through a sparse sub-sampling training technique. We show that this approach:

- generalizes across four well-defined circuit topologies of varying complexity,
- is accurate, as it obtains performances that meet at least 96.3% of tested target design specifications, while also considering post-layout simulations, which is an important part of the the design process,
- is up to 40× more run-time efficient compared to prior work, and
- is practical, as the failures in the tool reflect design constraints as opposed to anomalies in the framework.

## 2.1 Introduction to analog design and sizing

Analog design is an important part of the ASIC development process, as analog circuits are responsible for filtering, amplifying and retaining fidelity in continuous time-domain signals. Even though analog blocks in analog and mixed-signal (AMS) chips do not take as much area as their digital counterparts, the design effort that goes into creating these blocks is still dominated by analog circuits, primarily because they require significant manual design effort. Today's designers are confronted with higher simulation workloads that result from more corners and parasitic elements, tougher design challenges due to difficult-to-predict parasitic effects, and more layout effort because of generally higher device count and restrictive design rules [27]. In order to reduce time-to-market, it therefore becomes crucial to identify and automate time-consuming analog procedures in an efficient and accurate manner.

Analog sizing is one aspect of analog design, where a designer selects parameters for a given circuit topology such that it meets a known target specification. This process can be summarized as a guided multi-objective optimization problem, where designers are heavily involved in the iteration process and are responsible for understanding the tradeoffs between specification and parameters in order to find a feasible solution. Analog sizing is a difficult problem, however, primarily because it involves multiple schematic and layout simulations at different stages of the design process, and these often rely on designers to manually create the required circuit layout, which, when simulated, can take a long time. In addition, the traditional analog design process relies on designer intuition to drastically reduce the range of possible parameters in the search space. Even though this methodology can be manually scripted into a program, simulation results in newer technology nodes are difficult to predict due to enhanced parasitic effects. The relationship between parameters and specification can also vastly vary depending on the circuit topology, which ranges in complexity and variety, greatly increasing the number of tunable parameters in the design space, while also making any fixed circuit-specific methodology infeasible for a scalable analog sizing tool.

## 2.2 Prior analog sizing tools

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

Traditionally, expert circuit designers employ a knowledge-based analog circuit synthesis methodology [28, 29, 30, 31], where design considerations and parameter selection algorithms are manually codified or formulated as templates in an effort to encapsulate the designer's knowledge. This strategy, however, still requires circuit designers to be heavily involved in the sizing process, and though efficient in converging to a specification, it is not automated, and often does not consider parasitics explicitly as its effects are difficult to model. In addition it requires a large overhead in defining any new specification.

Equation-based methods [32] manually or automatically obtain constraint equations that are then optimized or solved. These solvers require very few, if any, simulation iterations, but involve manual equation formulations, do not consider parasitics, and only allow circuits with known dependencies to be characterized.

Bayesian optimization, genetic algorithms and their optimization-based variants have also been utilized extensively, primarily via stochastic sampling to simulate a certain population of design points which are either mutated based on a fixed cost function to produce new, potentially more fit points that are simulated again in the next iteration [17], or used to generate surrogate models [33]. Genetic algorithms, traditionally, require many simulations to converge and are not reliable, as the converged result heavily depends on how well the initial stochastic sampling reflects the true design space. In addition, they require the algorithm to be re-started whenever a change is made to the goal. Though Bayesian optimization solutions [33] are more sample efficient, formulating a Gaussian Process model is more expensive as the dimensionality of the data increases, meaning that this solution is potentially prone to issues in scalability. Several works exist that attempt to reduce the inefficiency of these algorithms, primarily via learning [34, 35, 16, 36]. [34], [35], and [36] do not, however, demonstrate results with layout parasitics, making it difficult to claim the reliability and practicality of these results. Authors in [16] demonstrate that their combined neural-network-boosted genetic algorithm can function on complex circuits with post-layout simulation following parasitic extraction (PEX), but the algorithm requires many PEX simulations to converge, making it difficult to scale to more complex circuits. Bayesian optimization methods [33], despite being more sample efficient, have not yet been proven with parasitic simulations.

Reinforcement learning (RL) has been applied to analog sizing as well, where an agent traverses the design space while trying to maximize a reward function. [37] explores initial results using this method, but only tests with NGSpice as the simulator, which is not reflective of the true complexity of state-of-the-art processes. [38] and [25] and use enhanced RL methods with recurrent or graph convolutional neural networks, respectively, that are trained using policy gradients. These algorithms do not consider layout effects and train the agent to meet only one target specification, requiring many circuit simulations to converge to other targets. [25] does however, show initial promise in using generalization to transfer information across topologies. [39] accelerates RL convergence by incorporating local constraint satisfaction while considering process corners, but fails to consider layout parasitics.

Despite the existence of automated layout-synthesis tools [40], prior sizing algorithms opt to use parasitics models [41, 42, 43, 44, 45, 46] to approximate layout effects without impacting simulation time. Approximate models show initial promise in predicting parasitic effects, but need to be further analyzed for accuracy and scalability on more complex circuits in newer and more parasitic-dominant technology nodes. Methodologies also exist that use lookup tables a-priori to simulate all relevant designs, but are time consuming and not scalable, as they require collecting a large number of layout simulations [47]. Authors in [44] propose a neural network metamodel that performs optimization in a sample efficient manner, requiring just two layout simulations to converge to a solution. Their methodology, however, requires a designer to manually create layout, and only tests on smaller-order circuits.

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

## 2.3 Introduction to machine learning

Machine learning (ML) is a growing field that studies the computational processes used by humans and machines [48], and is often used to solve complex tasks such as natural language processing, medical diagnosis, or speech recognition. ML encompasses a broad range of algorithms for classification, regression, optimization, recognition, clustering, and more, with new algorithms being developed as active research areas. Most ML methods take advantage of data by attempting to extract patterns within the data, or relationships between sets of data. The amount of data required, however, varies depending on the specific algorithm.

Determining whether ML is the right solution for a given task greatly depends on the type of problem, and is generally applied in cases where:

- deterministic rules cannot be codified easily for the task by using a simple algorithm, making it too difficult for a human to accurately code rules, and the
- existing solutions cannot scale in complexity to larger or more complicated data [49].

Traditional ML techniques are often seen as a black box; to the user the objective is reached but the process the algorithm uses to obtain it is unknown. Recently, researchers have looked into increasing the transparency and controllability of automated ML systems [50], allowing the efficacy, accuracy, interpretability and practicability of the algorithm to be analyzed. This analysis is especially important in tasks where reliability is crucial. It is the intent of this thesis to demonstrate application-specific study for a traditionally black-box ML algorithm, in the hopes of convincing the reader that ML methods can be understood effectively.

## 2.4 Reinforcement learning framework

Reinforcement learning (RL) is a class of techniques under machine learning that consists of an agent that interacts with its environment through a trial-and-error process that mimics learning in humans. It is a simulation-in-the-loop method, and relies on the ability to verify outputs from a true simulation source.

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

Unlike other ML techniques, reinforcement learning captures a designer's manual iterative approach to analog sizing. Crucially, RL attempts to learn the "how" of a problem through its training process, relying on a reward function to traverse the objective space in a combination of random search and reward-directed behavior. This not only aids in modelling complex behaviors, but also enables better transferability and generalization of the algorithm to changes in the environment. We make use of this fact in Section 2.4.5 to consider layout parasitic simulations, removing the need to have the algorithm train off of this expensive and difficult-to-collect data.

### 2.4.1 RL analog sizing setup

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

### 2.4.2 Trajectory generation

Upon reset, the circuit parameters are initialized to a fixed center point $\frac{K}{2}$. The deep neural network ($DNN$) then uses the observed performance $\mathbf{o}$ (created by simulating the circuit) and target specification $\mathbf{o}^*$, as well as current parameters $\mathbf{p}$ to output a probabilistic distribution that is sampled to determine whether to increment, decrement, or retain the same value for each circuit parameter (shown in Figure 2.1). The agent has $H$ total simulation steps to reach $\mathbf{o}^*$, where $H$ is nominally set to $K$. If the objective is reached before $H$ steps, the trajectory ends. We note that the fixed reset point is chosen strategically to ensure the agent reaches all $K$ points in the parameter space within the allocated $H$ steps.

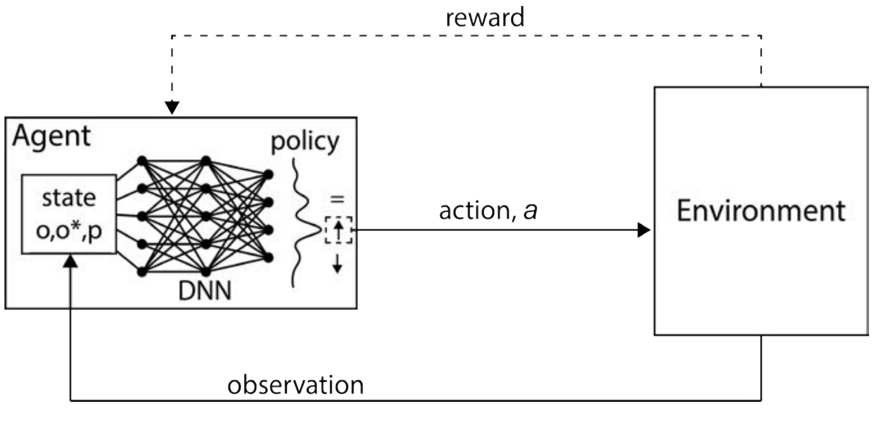


Figure 2.1: Reinforcement learning algorithm showing that the input to the network are the current performance, target specification, and current parameters, and the output is a probabilistic distribution that is sampled to determine whether to increment, decrement, or retain each circuit parameter.

### 2.4.3 RL agent

We choose a three-layer neural network, each with 50 neurons, to map between state and actions in the reinforcement-learning framework, whose system level diagram is shown in Figure 2.2. This architecture is selected primarily because its ease of training; other methods like recurrent networks are harder and more variable to train, requiring additional hand-tuning of hyperparameters.

As with most ML algorithms, the architecture details were tuned and selected iteratively for one circuit topology, based on the highest converged reward after training. In our studies, we find that three layers are deep enough to model the complexities of parameters to target specification in analog sizing, while not overfitting to specific regions of the design space. It is emphasized that this architecture, along with the RL-associated hyperparameters such as training batch size and horizon length, are agnostic to topology and technology on the circuits we have tested on, meaning that the network does not require hand-tuning for each application.

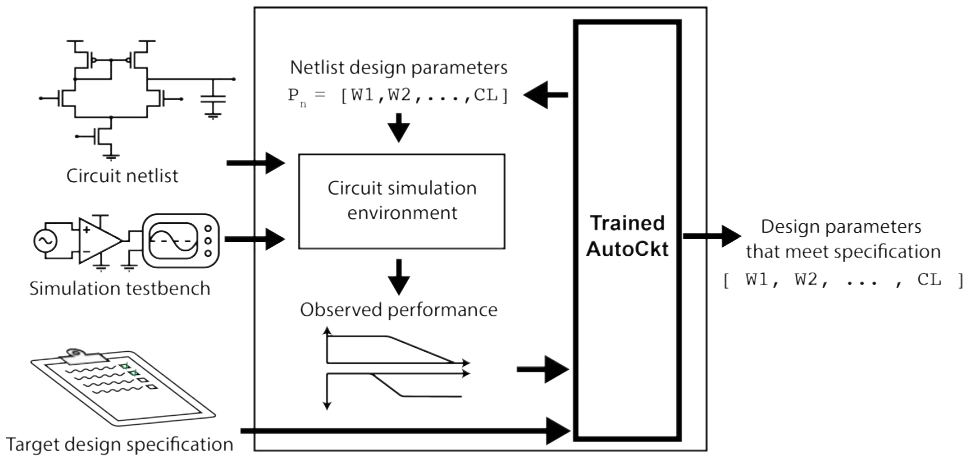

Netlist design parameters
P_n = [W1,W2,...,CL]
Circuit netlist
Circuit simulation environment
Simulation testbench
Observed performance
Target design specification
Trained AutoCkt
Design parameters that meet specification
[ W1, W2, ... , CL ]

Figure 2.2: Top level system diagram showing that AutoCkt takes as input a circuit netlist, its simulation testbench, and a target design specification to find parameters for the circuit such that it meets the specification.

### 2.4.4 Training with schematic simulations

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

$$r_o = min(\beta \frac{o - o^*}{o + o^*}, 0) \tag{2.1}$$

$$r = \sum_{o \in \mathbf{o}} r_o \tag{2.2}$$

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

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

$$R = \begin{cases} r, & \text{if } r < 0 \\ 10 + r, & \text{if } r = 0 \end{cases} \tag{2.3}$$

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

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

In the algorithm, if at any point a testbench is invalid, subsequent testbenches are not run and the target metrics from those simulations are set to $-1$, which is the smallest negative number in the bounded reward function in Equation 2.1. In addition to reducing simulation time, this technique also breaks down a multi-step optimization problem to easier sub-goals, allowing the agent to learn quicker. These validity tests are agnostic to the specific circuit type or topology, meaning that this methodology can be ported to many different designs. The algorithm also functions without this technique, as is demonstrated in one of the circuit topologies later in the results.

To train the RL agent, a batch of trajectories is collected with the schematic simulator, and the rewards, states (comprised of $\mathbf{o}$, $\mathbf{o}^*$, and $\mathbf{p}$) and actions (increment, decrement or retain each parameter) are used to update the weights of the neural network with gradient descent. Proximal policy optimization (PPO) [51] (summarized in Figure 2.3) is a policy gradient method that finds a precise local optima as opposed to other approaches like simulated annealing, which are approximate, and is used to formulate the objective function that maximizes the average accumulated reward. PPO is specifically known to be more stable and efficient during training by controlling how large the policy updates are at each iteration.

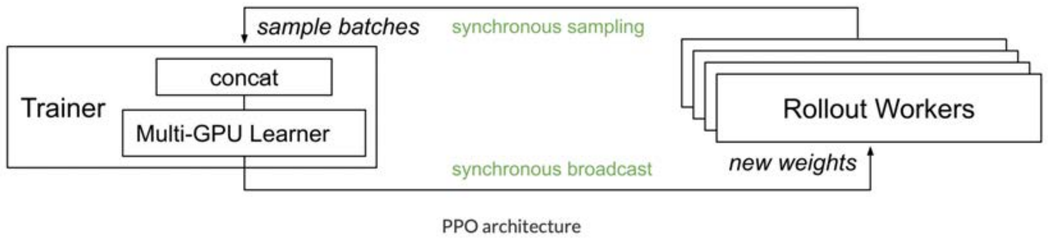


Figure 2.3: Proximal policy optimization framework setup using Ray [52].

Training sessions are conducted several times to ensure that AutoCkt, the reinforcement learning framework, is robust to variations in random seed. Training ends when the mean reward is greater than or equal to 0, signifying that the agent reaches the training target specifications for most trajectories. This training process is summarized in Algorithm 1. We use OpenAI Gym and Ray, a framework [52] for running distributed reinforcement-learning tasks, to efficiently implement the algorithm across a many-core CPU.

The framework and results on four example circuits across different simulation environments including NGSpice, Spectre and the Berkeley analog generator (BAG) [53] are described in more detail in the next section.

### 2.4.5 Deployment

During deployment, the trained agent generates trajectories when given unique target specifications, different from those in training. The generalization algorithm uses BAG to consider layout parasitics.

BAG allows designers to create, verify, and use process-portable circuit generators. The generator-based flow, shown in Figure 2.4, begins with a design script that notes the specific circuit topology, sizes of each device, and layout strategies. This information is then used as input to the schematic and layout generators, which takes a pre-defined schematic template and maps the input parameters to sizing of each component in the circuit or follows scripted layout strategies that can handle variations in sizes, respectively. Designers are responsible for creating comprehensive generator scripts such that the designs achieve DRC/LVS clean

**Algorithm 1:** Sparse sub-sampling training algorithm

```
1  Initialize $\mathbf{O}^* \leftarrow \mathbb{R}^{MxN}$
2  env $\leftarrow$ schematic simulator
3  while $\sum \frac{R}{N} < 0$ do
4      $Dataset \leftarrow []$
5      for $len(batchsize)$ do
6          $\mathbf{o}^* \leftarrow \mathbf{O}^*$
7          $t$ = Collect_Trajectory($\mathbf{o}^*$, horizon length, env)
8          $Dataset \leftarrow$ append($Dataset$, $t$)
9          $R \leftarrow$ append($R$, reward($t$))
10         PPO_Update_NN_Weights($R$,$Dataset$)
11     end
12 end
```

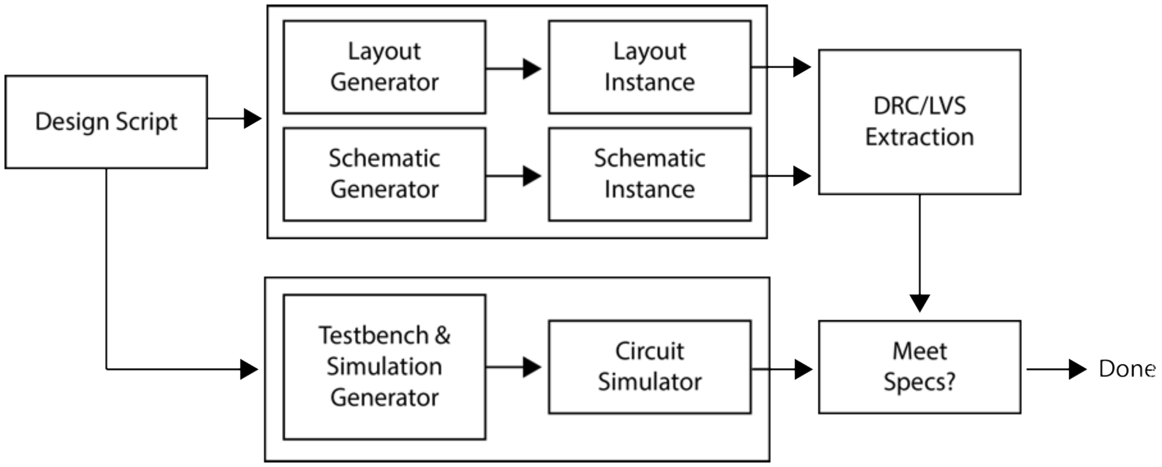


Figure 2.4: Diagram of generator-based design using BAG [55].

results for a wide range of parameter combinations. Testbench generators are also manually created to instantiate, run, and parse the results for each design. These testbenches integrate with existing CAD tool-chains that run and synthesize simulation results. Specifically, BAG integrates with a simulator tool called ADE-XL, which displays simulation data in hierarchical arrangement related to the circuit design, processes signal data, and provides calculator functions that allow for a variety of analyses [54].

We demonstrate the use of transfer learning, summarized in Algorithm 2, to show that an RL agent trained by running inexpensive schematic simulations is able to transfer knowledge to a different environment. This new environment, which then runs PEX simulations in BAG, is used to deploy the agent. The agent is given $2*H$ number of steps to meet the target in this

**Algorithm 2:** Deployment Algorithm Considering Layout Parasitics

```
1  $\mathbf{o}^* \leftarrow \mathbb{R}^M$
2  $env \leftarrow$ parasitic simulator
3  while $True$ do
4      $\mathbf{o}, \mathbf{p} \leftarrow$ step($\mathbf{o}, \mathbf{o}*, \mathbf{p}, env$)
5      $layout\_reward$ = calc_reward ($\mathbf{o}, \mathbf{o}*$)
6      if $layout\_reward \geq 0.0$ then
7          return $\mathbf{p}, \mathbf{o}$, (reached $\leftarrow$ True)
8      step_count += H
9      if $step\_count > H$ then
10         return $\mathbf{p}, \mathbf{o}$, (reached $\leftarrow$ False)
11 end
```

new parasitic simulation environment. In the results, there exist two forms of deployment: 1) where the trained agent is run on schematic-only simulations and 2) where the deployment is solely with PEX. Note that no re-training is required once the environment has changed to post-layout extraction.

## 2.5 Transimpedance amplifier results

AutoCkt's performance is demonstrated on a simple transimpedance amplifier (Figure 2.5) in $45nm$ BSIM predictive technology. The action space for each transistor consists of two separate parameters shown in array notation [start, end, increment]: width ($[2, 10, 2] * \mu m$) and multiplier ($[2, 32, 2]$). The feedback resistor action space consists of two parameters: number of resistors in series ($[2, 20, 2]$) and number of resistors in parallel ($[1, 20, 1]$). The fixed unit resistance is $5.6k\Omega$. The target metrics of interest are settling time ($[5, 500] * ps$), cutoff frequency ($[5.0e^8, 7.0e^9] * Hz$) and input-referred noise ($[100e^{-8}, 500e^{-6}] * V_{rms}$). The schematic has a fixed input and load capacitance of $50fF$ and $1pF$, respectively, and were chosen based on prior transimpedance amplifier specifications in this technology. For this circuit topology, we do not employ the simulation-reduction technique mentioned in Section 2.4.5, to demonstrate the functionality of the framework without this feature. Figure 2.6 shows the mean episode reward over time increasing to greater than zero after training has completed, meaning that the agent has learned to reach the positive reward state across multiple target specifications.

The trained agent is then deployed on 500 randomly chosen target specifications in the range specified above, summarized in Table 2.1. The results show that AutoCkt has a 25.1× speedup compared to a vanilla genetic algorithm, labelled required simulation count (RSC) in Table 2.1. Additionally, it is able to generalize to 97.4% of the design space, meeting 487 targets. Note the genetic-algorithm efficiency is determined by the best result obtained when sweeping initial population sizes and several target specifications.

Table 2.1: Comparison of prior analog sizing methods with required simulation count to converge and number of specifications reached versus tested: transimpedance amplifier.

| Metric | Required Simulation Count | # Specs Reached / # Specs Tested |
|---|---|---|
| Genetic Algorithm | 376 | N/A |
| This Work | 15 | 487/500 |

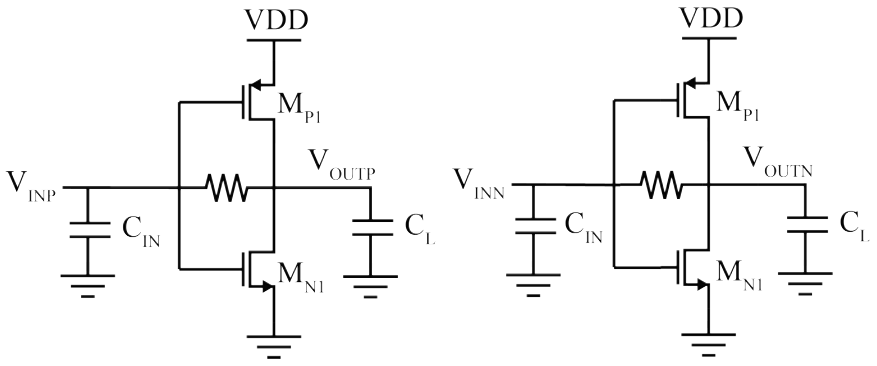


Figure 2.5: Simple transimpedance amplifier schematic.

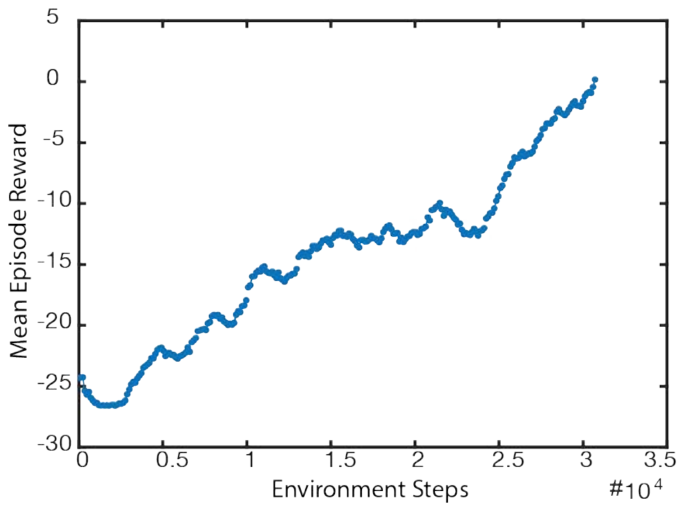


Figure 2.6: Mean episode reward for transimpedance amplifier.

## 2.6 Two-stage amplifier

AutoCkt is now tested on a more complex, yet common two-stage operational amplifier circuit with frequency compensation (Figure 2.7) in $45nm$ BSIM predictive technology. The design of this topology is well studied, with the equations used in AC simulations to obtain a target gain ($A_v$), unity-gain bandwidth ($UGBW$), and phase margin ($pm$) dictating the values for each parameter in the circuit.

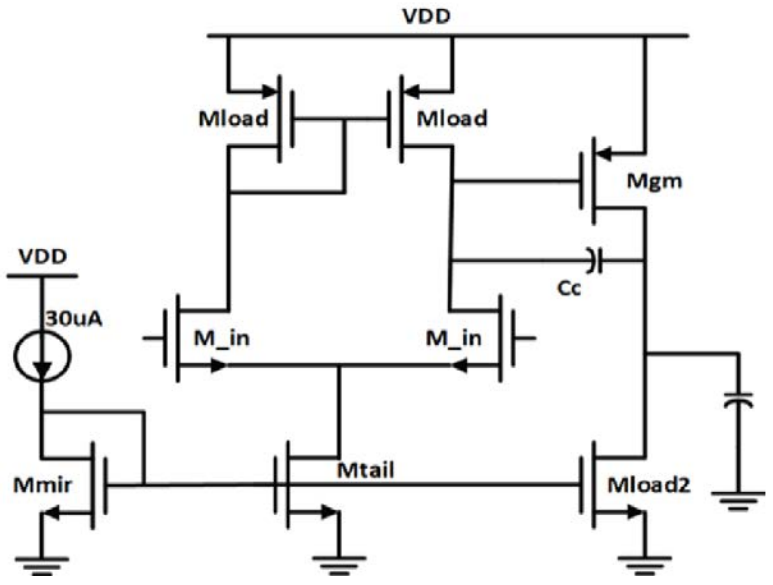


Figure 2.7: Two-stage operational amplifier schematic.

### 2.6.1 Manual design procedure

Specifically, the compensation capacitor ($C_c$) is chosen by considering the phase margin requirement, which for a nominal 60° requirement is equal to:

$$C_c = \frac{\sqrt{3}G_{Min}}{\omega_2 A_o} = \frac{\sqrt{3}g_{Min1,Min2}}{g_{Mgm}/C_L} \tag{2.4}$$

where $G_{Min}$ is first stage transconductance, $\omega_2$ is the second pole location assuming the load capacitance is dominant, and $A_o$ is the gain at the 60° phase margin.

The unity-gain frequency is then given by

$$UGBW = \frac{G_{Min}}{2\pi C_c} \tag{2.5}$$

With Equations 2.4 and 2.5, the transconductance of the $M_{gm}$ transistor can be obtained, with which $M_{gm}$'s drain current, $I_{D,Mgm}$, can be calculated. Assuming nominal overdrive voltage through $M_{gm}$, chosen based on output voltage swing requirements,

$$I_{D,Mgm} = \frac{g_{Mgm} V_{ov,Mgm}}{2} \tag{2.6}$$

The size of $M_{gm}$ is then:

$$(\frac{W}{L})_{M_{gm}} = \frac{g_{Mgm}^2}{2 I_{D,Mgm} \mu_p C_{ox}} \tag{2.7}$$

$M_{load2}$ is calculated using the same overdrive voltage requirement as before:

$$(\frac{W}{L})_{M_{load2}} = \frac{2 I_{D,Mgm}}{\mu_n C_{ox} V_{ov,load2}^2 (1 + \lambda_n V_{out})} \tag{2.8}$$

The gain of the second stage, $A_{v2}$, is then:

$$A_{v2} = g_{m,Mgm}(r_{o,Mgm} || r_{o,load2}) \tag{2.9}$$

To design the first stage of the amplifier, a nominal bias current is used to ensure that a power specification is met, which then dictates the tail current of the differential pair. With these requirements, $g_{Min1,Min2}$ is now:

$$g_{Min1,Min2} = A_{v1}(\lambda_n + \lambda_p) I_{D,Min} \tag{2.10}$$

The compensation capacitance can then be calculated using Equation 2.5. The sizes of the $M_{in}$ transistors is:

$$(\frac{W}{L})_{M_{in}} = \frac{g_{Min1,Min2}^2}{2 I_{D,Min} \mu_n C_{ox}} \tag{2.11}$$

The current mirror load transistors' ($M_{load}$) gate-to-source and by KVL the drain-to-source voltages must be equal to the gate-to-source voltage of $M_{gm}$ in order to be in saturation. Therefore the sizes can then be derived by:

$$(\frac{W}{L})_{M_{load}} = \frac{2 I_{D,Min}}{\mu_n C_{ox} V_{ov,Mgm}^2 (1 + \lambda_p (V_{ov,Mgm} + V_{t0}))} \tag{2.12}$$

The $M_{tail}$ transistor is sized by considering $M_{load2}$, such that they receive the correct currents through their branches:

$$(\frac{W}{L})_{M_{tail}} = (\frac{W}{L})_{M_{load}} \frac{I_{D,tail}}{I_{D,load2}} \tag{2.13}$$

The width to length ratio of $M_{mir}$ is chosen to be a multiple of $M_{tail}$ and $M_{load2}$, to correctly provide the required current through each of these branches from the current source.

Initially, the lengths of the devices are selected to be minimum channel length to minimize area.

These equations dictate the detailed procedure used by analog circuit designers to obtain a target specification. Despite this, however, successfully sizing this operational amplifier involves iteration, with the procedure only providing an initial starting point which often does not meet its target in the first iteration. We thus move to automate this sizing process with reinforcement learning, and discuss the framework setup in the next section.

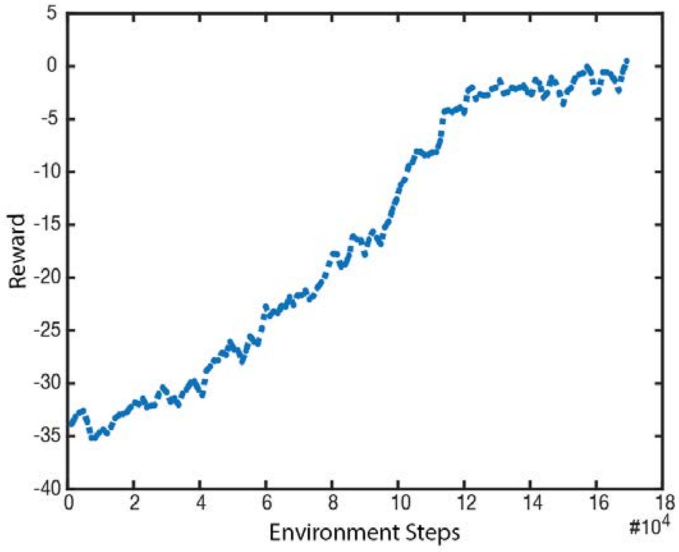


Figure 2.8: Mean episode reward for two-stage amplifier.

### 2.6.2 Operational amplifier reinforcement learning setup

The operational amplifier topology is shown in Figure 2.7. The action space for every transistor width in the schematic is $[1, 100, 1] * 0.5\mu m$, chosen by nominally sampling around typical transistor sizes for each parameter. The compensation capacitor ranges from $[0.1, 10.0, 0.1] * 1pF$. The design specifications of interest are gain ($[200, 400] * V/V$), unity-gain bandwidth ($[1.0e^6, 2.5e^7] * Hz$), phase margin ($[60.0]*^\circ$), and power (as a measure of bias current, $[0.1, 10] * mA$), with the load capacitance fixed to $10pF$. The total action space size is $10^{14}$ possible values, making random generation of parameters to meet the target design specification infeasible. The agent is allowed a horizon length of 30 simulation steps to converge. The mean reward over total environment steps is shown in Figure 2.8.

Note that this circuit topology is trained without the sequential sizing methodology, and though the agent takes $10^4$ steps to reach the stopping condition, the amount of time to

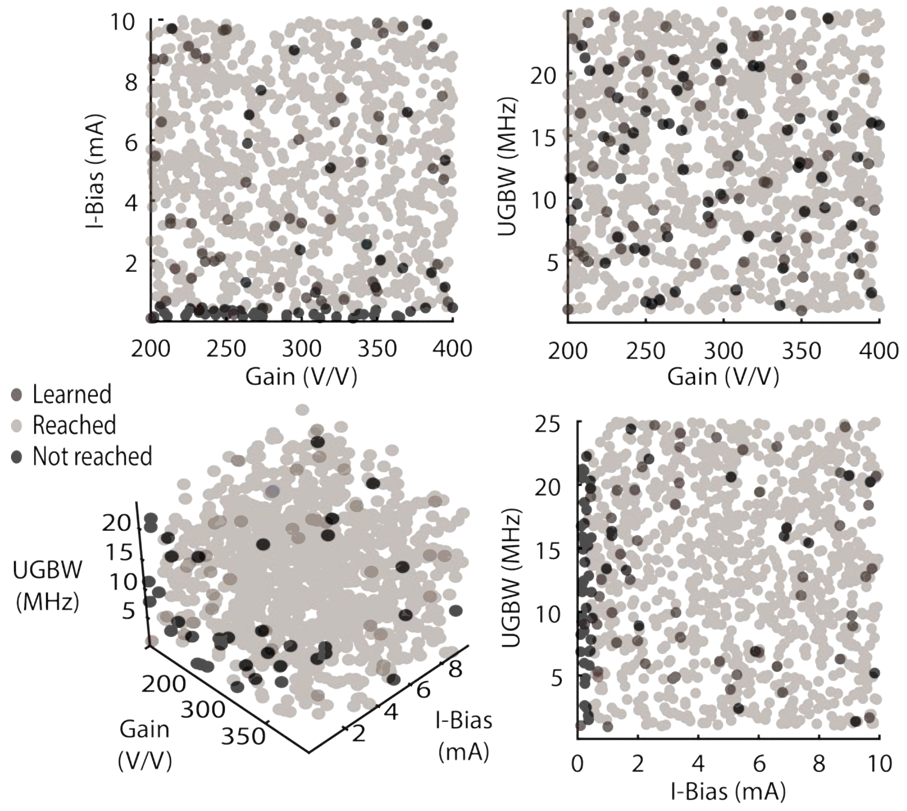


Figure 2.9: Distribution of obtained or learned, reached, and unreached target design specifications. Bottom left shows the 3D plot with three of the four design metrics. The rest of the plots show differing metric combinations to visually demonstrate which design points are not met.

complete is expedited due to Ray's distributed compute capabilities [52]. Thus the wall clock time is just 1.3 hours on a 8-core CPU machine.

The trained agent is deployed on 1000 randomly generated target design specifications it has never seen before, in the range specified during training. The results of which target specifications are unmet and met are shown in a 3D plot (Figure 2.9, phase margin is excluded because it only has a lower-bound requirement). The distribution of points in Figure 2.9 show that the unreached design points fall along a vertical region where bias current is very low. It can then be hypothesized that these points are indeed unreachable given the power requirement. Looking at the converged design specifications for these unreached points, the agent attempts to meet the gain and bandwidth requirement while minimizing for power, similar to how a circuit designer approaches this problem. Table 2.2 shows several agent-obtained design compared to that of an expert, and shows that it achieves comparable performances. Table 2.2 includes the efficiency as well, calculated by $f_t C_L / i_{bias}$.

Authors in [56] derive that for a two-stage amplifier, the estimated efficiency figure of merit (FOM) should be in the hundreds of $MHzpF/mA$, which means that the agent-obtained performances are feasible and valid in the context of circuits as they fall within this range.

The comparison table is shown in Table 2.3. Note that the comparison also includes a random RL agent taking steps in the environment to illustrate design-space complexity. The results demonstrate that AutoCkt reached 963 of the 1000 target design specifications, generalizing by a factor of 20× compared to the specifications seen during training. Of those points it does reach, the average number of simulation steps it takes is just 27, which is near 40× faster than a traditional genetic algorithm.

## 2.7 Low-dropout voltage regulator

Voltage regulators are crucial for SoCs, as the supply voltage powering other areas of the chip must be stable, and ideally noiseless, for proper functionality. Low dropout (LDO) voltage regulators, in particular, are used for their high power efficiency and stability to changes in load. We discuss the general sizing procedure for a traditional LDO topology, and test the sizing framework on the LDO in 14$nm$ technology.

### 2.7.1 Manual design procedure

A topology similar to Figure 2.10 is selected. The manual design procedure, outlined by [57], involves sizing the error amplifier, the current-controlled source ($m_{pass}$), and the corresponding current mirrors for gain, phase margin, line and load regulation, noise and power-supply rejection ratio requirements.

Specifically, the loop gain, $A_{LG}$ can be calculated by breaking the feedback loop, assuming a small-signal resistance $R_L$ to the load:

$$A_{LG} = A_1 g_{m1}[R_L||(R_{f1} + R_{f2})]\frac{R_{f2}}{R_{f2} + R_{f2}} \tag{2.14}$$

where $A_1$ is the gain of the amplifier in Figure 2.10. $V_{out}/V_{in}$ is then calculated to analyze how $A_1$ affects power-supply rejection ratio (PSRR):

$$\frac{V_{out}}{V_{in}} = \frac{g_{m1}[R_L||(R_{f1} + R_{f2})]}{1 + A_1 g_{m1}[R_{f2}||(R_{f1} + R_{f2})]\frac{F_2}{R_{f1}+R_{f2}}} \approx (1 + R_{f1}/R_{f2})\frac{1}{A_1} \tag{2.15}$$

Load regulation measures the change in output voltage over the change in output load current, which is essentially the output impedance, $R_{out}$, of the circuit:

$$R_{out} \approx (1 + \frac{R_{f1}}{R_{f2}})\frac{1}{g_{m1}A_1} \tag{2.16}$$

Table 2.2: Agent-obtained performances and an expert design for the two-stage amplifier with load capacitance of $10pF$.

| Metric | Agent Performance 1 | Agent Performance 2 | Agent Performance 3 | Expert Design |
|---|---|---|---|---|
| UGBW ($MHz$) | 10.2 | 14.2 | 13.8 | 10.66 |
| Gain ($V/V$) | 356 | 289 | 298 | 314 |
| PM (°) | 62.2 | 59.8 | 58.3 | 60.83 |
| I-Bias ($\mu A$) | 146 | 283 | 195 | 158 |
| Efficiency ($MHzpF/mA$) | 690 | 502 | 708 | 689 |

Table 2.3: RSC and number of specifications reached versus tested: two-stage op amp.

| Metric | Op Amp RSC | TIA RSC | # Specs Reached / # Tested |
|---|---|---|---|
| Genetic Algorithm | 1063 | 376 | N/A |
| Random RL Agent | N/A | N/A | 38/1000 |
| This Work | 27 | 15 | 963/1000 |

Both Equations 2.15 and 2.16 suggest that the operational-amplifier loop gain should be maximized for the best PSRR and load regulation, respectively. To model output noise for the circuit, the following equation is used:

$$\overline{V_{n,out}^2} = (1 + \frac{R_{f1}}{R_{f2}})^2(\overline{V_{nA}^2} + \frac{\overline{V_{nM}}}{A_1^2}) \tag{2.17}$$

where the noise of $m_{pass}$ is $\overline{V_{nM}^2}$, and the noise of the amplifier is $\overline{V_{nA}^2}$. Once again, increasing $A_1$ leads to a better noise figure. Having higher amplifier gain however leads to increases in power and sizes of the devices. In this study, we include the power target to successfully constrain and model this behavior.

The frequency response of the system can be modeled by replacing the gain $A_1$ with $A_0/(1 + s/\omega_0)$, where $w_0$ is the pole frequency:

$$\frac{V_{out}}{V_{in}} \approx (1 + R_{f1}/R_{f2})\frac{1}{g_{m1}A_0}(1 + \frac{s}{\omega_0}) \tag{2.18}$$

These equations, as with the previous two-stage amplifier circuit, are used as initial starting points in an iterative sizing procedure.

## 2.7.2 LDO target metrics and action space

Unity-gain bandwidth ($UGBW$), phase margin ($pm$), noise, output-voltage stability with load variation ($V_{out}$), and power-supply rejection ratio (PSRR) are typically used in the specification for an LDO. For the initial experiment, we select $UGBW$, $pm$, noise, and $V_{out}$ as hard constraints, and use the remaining metrics for performance analyses. It is emphasized that incorporating these or any other additional constraints into the framework is relatively simple.

The action space includes the number of fingers $[2, 12, 1]$, multiplier $[1, 10, 1]$, number of fins $[2, 12, 1]$ and length $(14nm, 100nm, 200nm)$ for every transistor, resistance $[2, 3, 0.1] * k\Omega$, and capacitance $[2, 15, 1] * fF$ values.

For this circuit, we employ the sequential simulation methodology introduced in Section 2.4.4. In addition, information from several intermediary values and constraints that are not explicitly part of a performance specification are given to the agent as part of the state.

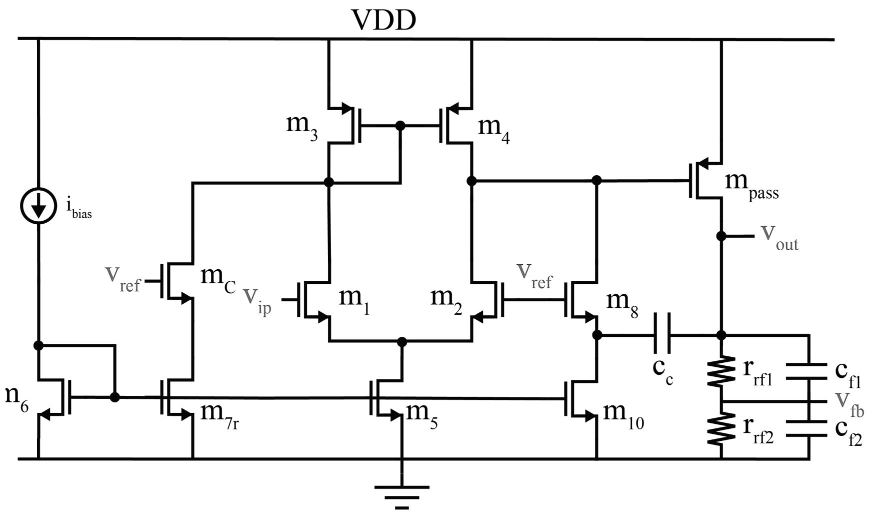


Figure 2.10: LDO schematic.

### 2.7.3 Training and deployment results

Figure 2.11 shows the mean reward for each training iteration reaching the stopping condition. we include both the baseline RL agent, as well as the proposed work that incorporates sequential testbench simulations and intermediary values, both of which run the same 100 targets. The baseline test takes 3.2× more samples to train and does not yet reach the stopping condition.

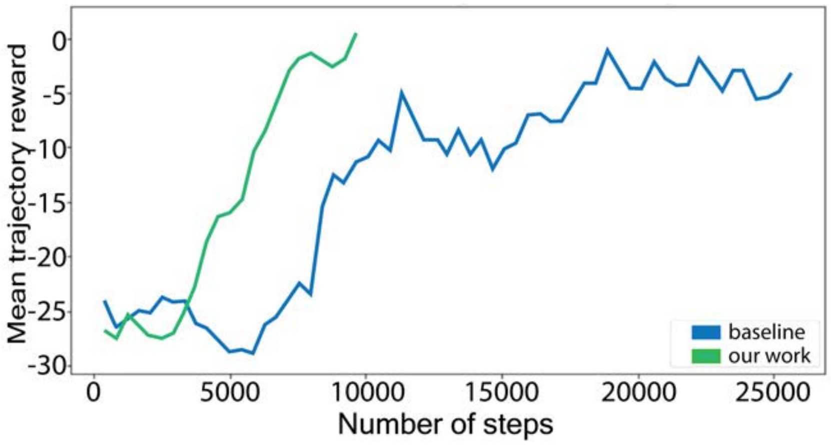


Figure 2.11: Mean episode reward for LDO.

Table 2.4: Training and deployment number of specifications reached versus tested: LDO.

| Metric | # Reached / # Tested: Training | # Reached / # Tested: Deployment |
|---|---|---|
| Our algorithm | 75/100 | 220/300 |

The number of target specifications the agent meets during training and deployment (with 300 unique targets) are tabulated in Table 2.4. Both percent accuracy values are somewhat low, but very similar, meaning that the agent does not overfit to the training targets. The low accuracy is explained in Section 2.7.4, where we posit that the unreached target specifications are not possible for this topology due to technology and parameter constraints.

### 2.7.4 Practicability and interpretability analyses

In this section, we analyze the results to understand:

- why the agent does not meet certain target specifications,
- the distribution of circuit performances the agent can achieve for this topology,
- the validity of the agent's designs from a circuits perspective,
- how the agent's design compares to an expert-designed LDO, and
- how the algorithm compares to prior state-of-the-art LDO sizing tools.

We aim to demonstrate the practicability, accuracy, and interpretability of outputs produced using this RL framework. Establishing these characteristics is crucial in determining the practicality of an analog sizing tool.

#### Reached and unreached target specifications

We analyze which target specifications the agent reaches and which are unreachable. Figure 2.12 shows 2D scatterplots of unity-gain bandwidth, noise, and output voltage, color-coded based on whether they are met during deployment. Phase margin is not depicted here, as the agent is trained on a single lower-bound constraint of $> 60°$. The results show a distinct distribution of unreached target specifications. In particular, aiming for an output voltage below $0.66V$ nearly always causes failure. This follows intuition, as the output voltage is determined by the operating point of the output node of the circuit. This fixed value is at $0.74V$, constrained by the technology node and topology, as well as the driving current source. There is a distinct failure pattern for low noise targets as well.

Overall, we find that noise and output voltage are critical in determining success or failure of the agent, and the agent generally meets the unity-gain bandwidth and phase-margin requirements. We posit that the targets the agent is not able to meet define a performance boundary for this topology, constrained by technology node and input parameter ranges.

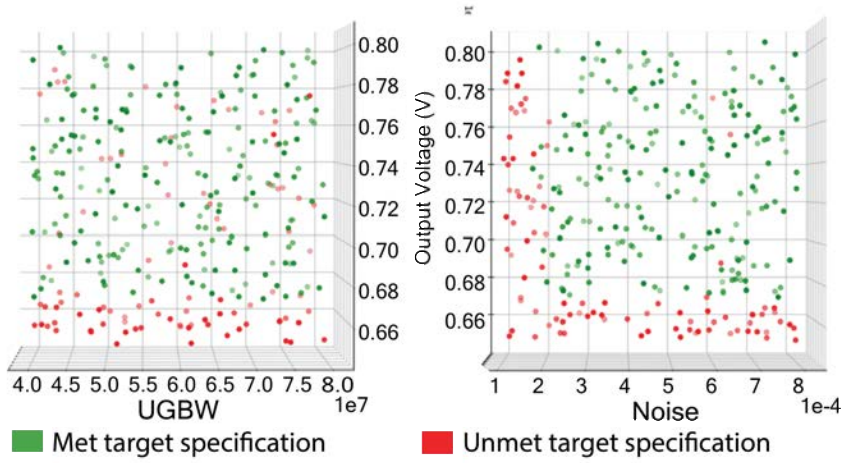


Figure 2.12: Met and unmet target specifications for LDO.

Table 2.5: Obtained performances by agent for LDO.

| Metric | Mean | Standard Deviation |
|---|---|---|
| Noise | $185\mu V_{rms}$ | $11.4\mu V_{rms}$ |
| Unity-gain bandwidth | $74.1MHz$ | $9.7MHz$ |
| Output voltage | $0.757V$ | $1.1mV$ |
| Phase margin | $67.0\circ$ | $4.2\circ$ |

## Unique obtained performances

We compare target specifications with obtained performances to determine whether the agent is learning to optimize the design space, or is converging to one known good design and relying on reaching that point during deployment. Table 2.5 summarizes the mean and standard deviation of the metrics obtained by the agent. This shows that the agent does in fact obtain a range of performances in the design space for unity-gain bandwidth, phase margin, and noise. The output voltage varies as expected, due to the fixed nature of the operating point. In total the agent reaches 215 unique performances. It can then be confirmed that the agent is indeed optimizing the design space based on the target specification given to it.

We also look at the difference between the distribution of reached performances by the agent versus the target specification given to it, summarized in Figure 2.13. Visually, the distributions are very different, and the obtained performances appear closely clustered in one corner of the design space. This plot shows the importance of target selection in the agent's sizing process, as the randomly generated targets often do not reflect the true feasibility of possible performances once topology and parameter constraints are considered. Despite this however, the agent is still able to successfully converge to meet target specifications.

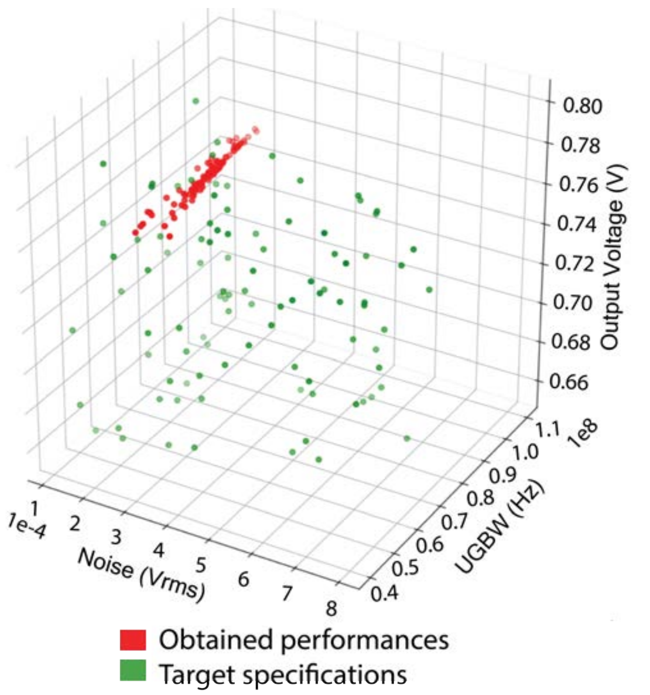


Figure 2.13: Obtained performances by agent versus target specifications.

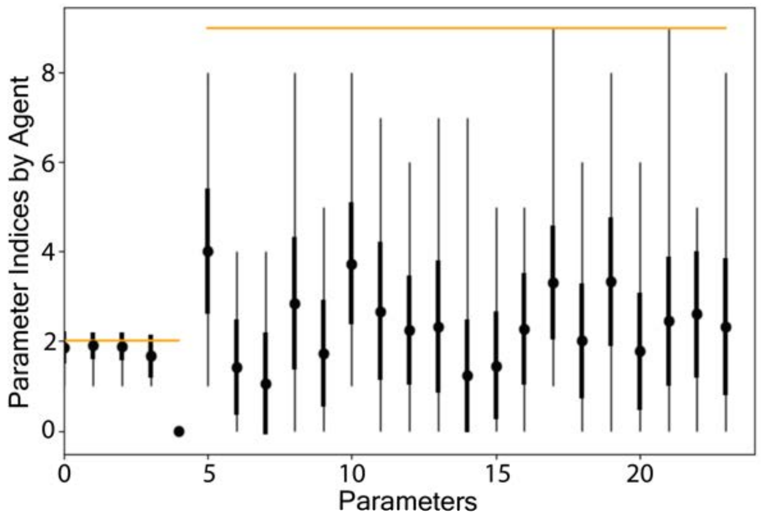


Figure 2.14: Obtained parameters by agent for reached performances.

Figure 2.14 shows how the circuit parameters vary across the different designs produced by the agent. This is important as we want to ensure that the agent is going through a full optimization process in exploring the parameter space, as opposed to finding one set of parameters that satisfy many different targets. The histogram shows the mean, minimum and maximum, along with standard deviation for of the parameters tuned by the agent (represented as indices of an array). The orange lines show the maximum possible value for each parameter. From these results, it is ascertained that the agent explores a wide range of parameter values, and utilizes most of the range of each of the parameters to obtain performances.

## Validity of Obtained Performances

We now analyze the validity of the agent's designs from a circuit's perspective, which dictates whether the tool can be practicable and accurate. There are several ways that designers check that results are valid in the context of circuits. One method is to examine transistor operating points to ensure that devices operate in the correct regime. For this topology, we measure the voltage headroom of the error amplifier devices, which should be greater than $0V$. Figure 2.15 shows maximum, mean, and minimum headroom values for an expert-designed LDO with corner simulations, and that the agent's corresponding headroom values fall within this feasible range.

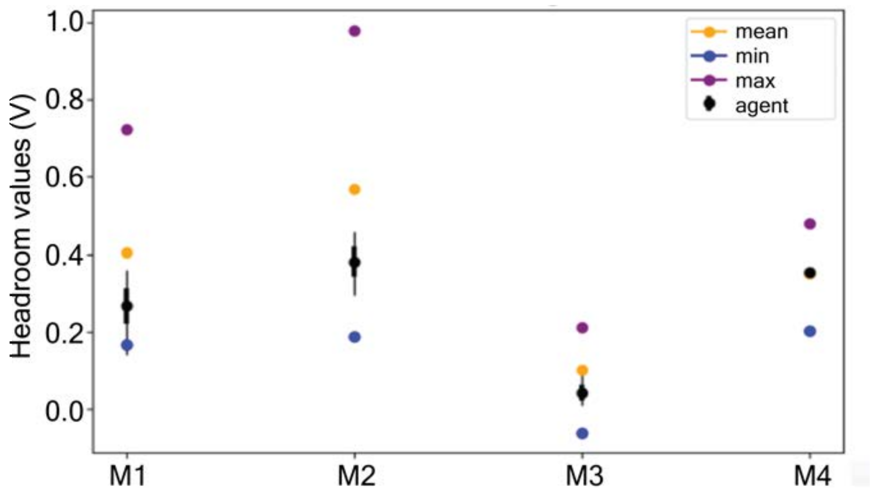


Figure 2.15: Agent-obtained headroom values for several devices.

Note that the agent converges to slightly lower mean headroom values compared to the expert design is not of concern, primarily because it shows $tt$ corner simulations only whereas the comparison is made to an expert's singular design while running nominal as well as $ff$, $ss$, $sf$,and $ss$ corners. We later compare the agent's $tt$ performance with an expert design in the same corner for a fairer comparison. Note that a headroom constraint for these transistors can be given to the agent during training if necessary.

### Comparison to Expert-Designed LDO Topology

Table 2.6: Agent versus expert: LDO.

| Metric | Expert Design | Agent | % Diff. |
|---|---|---|---|
| UGBW | $69MHz$ | $73.8MHz$ | 6.45% |
| Noise | $188\mu V_{rms}$ | $188\mu V_{rms}$ | 0% |
| $V_{out}$ | $0.756V$ | $0.757V$ | 0.1% |
| Power | $871\mu A$ | $860\mu A$ | $-1.27\%$ |
| Phase Margin | $69.7\circ$ | $72.9\circ$ | 4.54% |
| Load current range | $1\mu A - 1mA$ | $1\mu A - 1mA$ | 0% |

We now directly compare the agent's obtained performance against an expert's design when given the same target specification. The targets are taken from a practical, real-world application, and the expert design was manufactured in silicon. Table 2.6 shows the percent difference between each metric. In this case, both are running the *tt* corner. We find that the agent not only reaches comparable performances, but higher unity-gain bandwidth and phase margin, for lower power compared to the expert.

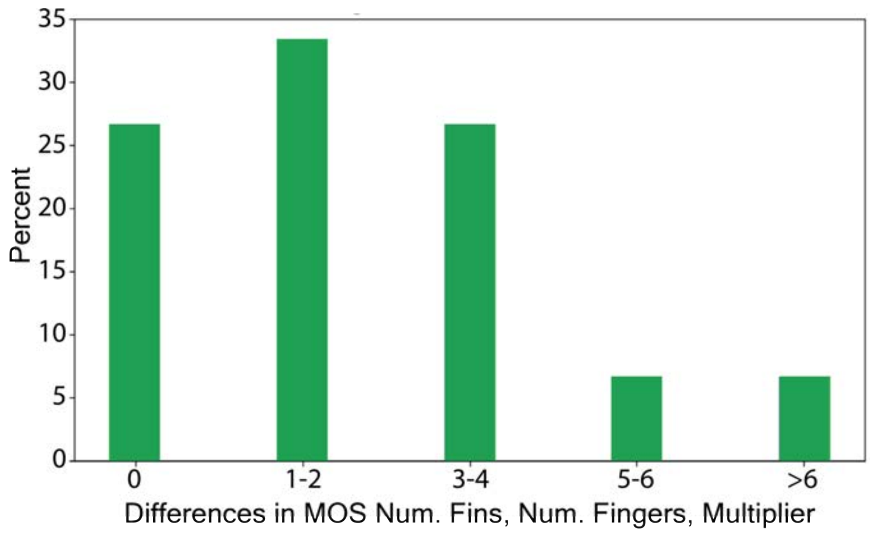


Figure 2.16: Comparing agent versus expert parameter values.

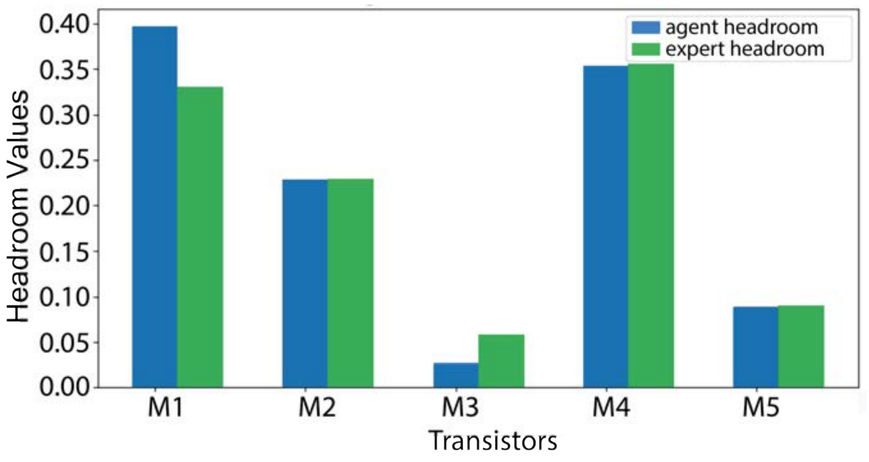


Figure 2.17: Headroom values for expert and agent designs.

We also assess what parameters are used by the agent for this design, and compare against the expert designer. Values for MOS number of fingers, fins, and multiplier between the agent the expert have a Pearson's correlation coefficient of 0.996. Figure 2.16 shows a histogram of the differences in parameters between the designs. 60% of the data falls within just one to two values of each other, and only 5% of the devices have a difference greater than six. The total root-mean-squared error between both designs is 9.4. Even though the obtained performance of the agent is different from that of the expert, sanity checking that the parameters are close to each other ensures some amount of validity in the design that the agent produces.

Lastly, we compare the voltage headroom for the error amplifier devices, shown in Figure 2.17. The agent's design reaches positive headroom values that are comparable or higher than the expert's design. Instances of lower headroom for the agent (e.g. device M3) can be improved, if need be, by either incorporating process-corner variation in the simulation environment or introducing a voltage headroom constraint to the agent. For this analysis, headroom is only used to confirm that all devices are in a valid mode of operation.

### Comparison to prior work

We compare the results of the algorithm with three automated LDO-sizing tools, two of which utilize mathematical constraint models, and one that uses reinforcement learning. The results are summarized in Table 2.7. Compared to prior work, we show an automated approach to sizing an LDO, that takes very few required simulations. In addition, we demonstrate that this algorithm reaches many design specifications on a new technology node.

Table 2.7: LDO comparison table.

| Characteristic | [58] | [59] | [25] | This Work |
|---|---|---|---|---|
| Algorithm | Geometric program | Linear program | Graphs+RL | RL |
| Automated | No | No | Yes | Yes |
| Technology | $0.18\mu m$ CMOS | $0.18\mu m$ CMOS | $180nm$ TSMC | $14nm$ FF |
| # tunable params | 23 | 24 | 29 | 24 |
| # tested specs | 1 | 1 | 1 | 300 |

## 2.8 Two-stage operational transimpedance amplifier with negative $g_m$ load

We now test the algorithm on an expert-designed two-stage operational amplifier with negative $g_m$ load in 16nm TSMC technology. The manual design procedure is outlined in-depth in Section 3.2, and includes AC, transient and noise metrics.

### 2.8.1 Schematic results

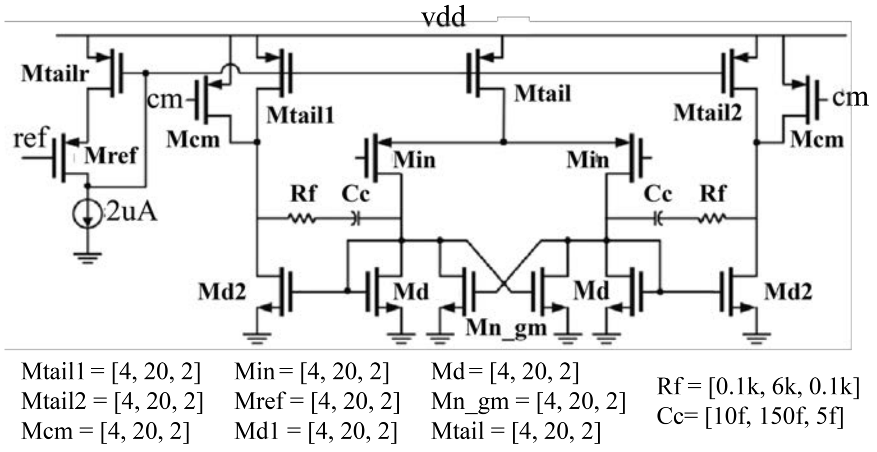


Figure 2.18: Schematic and action space for two-stage op amp with negative $g_m$ load.

This circuit topology in Figure 2.18 contains negative $g_m$ and diode-connected loads in the first stage, thereby having positive feedback and making the circuit more challenging to design. The action space ranges are summarized in the schematic; in total the order of complexity is $10^{11}$ parameter combinations. The range for each target metric is chosen around an actual target design specification determined by an expert: gain ($[1, 40] * V/V$), unity-gain bandwidth ($[1.0e^6, 2.5e7] * Hz$), and phase margin ($[60, 75]*°$). Initial results

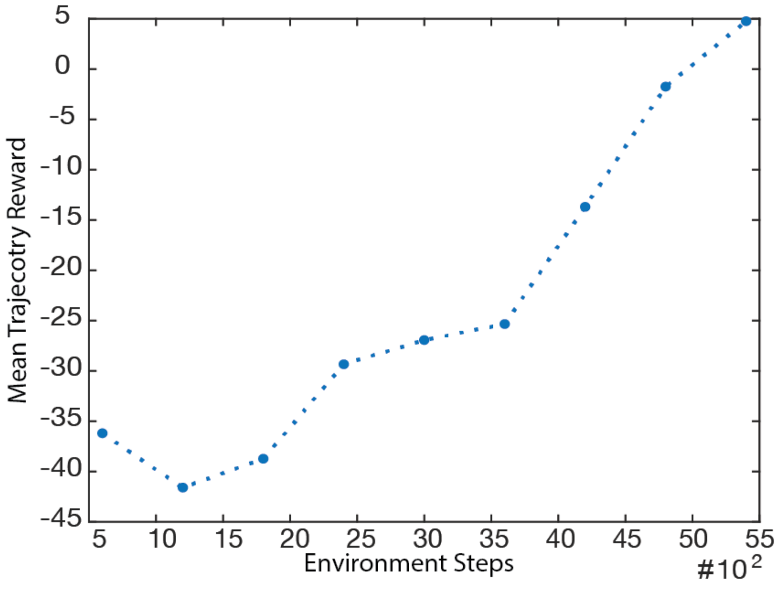


Figure 2.19: Mean episode reward over environment step for negative $g_m$ operational amplifier.

only include AC testbench metrics; we demonstrate the results for a more complete target specification in the next chapter.

The mean episode reward per number of steps is depicted in Figure 2.19, and shows that the agent successfully reaches the stopping condition by obtaining many target specifications. Figure 2.20 shows the deployment results for 500 randomly generated target specifications after training the agent. Note that there are no unreached specifications. The comparison table presented in Table 2.8 shows very similar results compared to prior circuit topologies, with 40.6× faster convergence to a target specification compared to a traditional genetic algorithm, taking on average just 10 simulations to converge to a solution.

Table 2.8: RSC and target specifications reached versus tested: two-stage operational amplifier with negative $g_m$ load.

| Metric | Op Amp RSC | # Specs Reached / # Specs Tested |
|---|---|---|
| Genetic Algorithm | 406 | N/A |
| Random RL Agent | N/A | 4/500 |
| This Work | 10 | 500/500 |

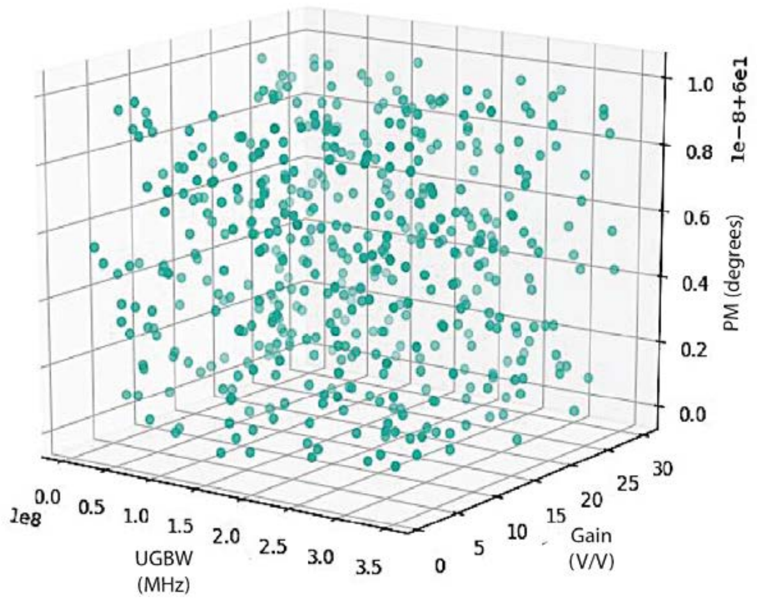


Figure 2.20: Distribution of reached target design specifications for the operational amplifier with negative $g_m$ load. Note that this example does not contain any unreached objectives.

### 2.8.2 Layout simulation results

To demonstrate AutoCkt's functionality once layout parasitics are considered, the algorithm is tested with a BAG designed two-stage amplifier with negative $g_m$ load. The target specifications are randomly chosen within the same range as the schematic-trained agent with the exception of phase margin, where we only enforce a minimum requirement of 60°. In these tests, we find that training on a range of phase margins, as opposed to a single lower bound of 60°, resulted in a better generalization. This is likely due to the agent benefiting from more exploration of the design space.

In general, compared to schematic counterpart, the transferred agent takes about 2.3× longer to converge to a design that meets the target specification (shown in Table 2.9) due to the addition of layout parasitics. Figure 2.21 shows a histogram of 50 design points that calculate the average percent difference across each design specification between PEX and schematic simulation. We posit that the agent learns the intuitive tradeoffs between parameters and design specifications as well as the best actions to take to move towards a goal, and that these relationships hold when considering layout parasitics despite potentially large amounts of difference between the schematic and PEX simulations.

Table 2.9 shows that running a vanilla genetic algorithm is too sample inefficient. AutoCkt is also compared to the combined machine learning and genetic algorithm [16] and show that the sample efficiency of AutoCkt is 9.56× greater than the prior state-of-the-art. Running on a single core CPU, the algorithm takes just 1.7 hours to complete. The algorithm is run on 40 randomly generated target design specifications, and AutoCkt is able to to obtain 40 LVS-passed designs in under three days, with no parallelization.

Table 2.9: RSC and target specifications reached versus tested: two-stage operational amplifier with negative $g_m$ load with layout parasitics.

| Metric | RSC | # Specs Reached / # Specs Tested |
|---|---|---|
| Genetic Algorithm | N/A | N/A |
| Genetic Algorithm +ML [16] | 220 | N/A |
| AutoCkt Schematic Only | 10 | 500/500 |
| AutoCkt PEX | 23 | 40/40 |

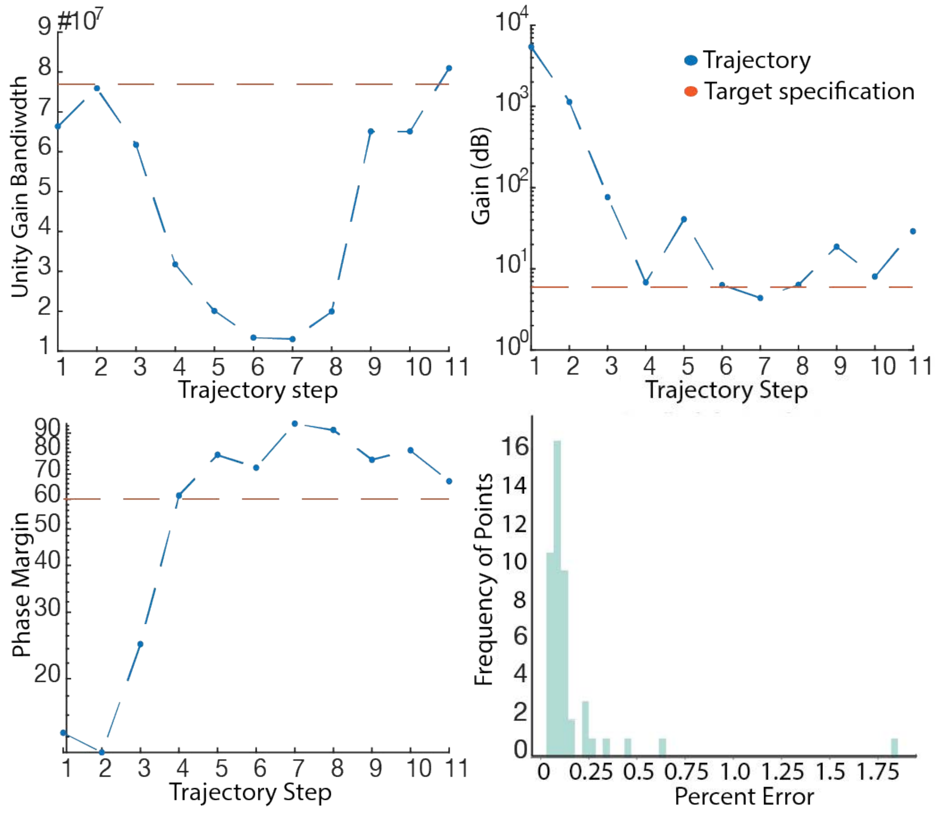


Figure 2.21: Top left, top right, and bottom left figures show a sample trajectory for the transferred agent attempting to reach one target design specification. Bottom right shows a histogram plotting difference between schematic and layout simulation.

## 2.9 Analysis and conclusion

In this chapter, we introduce the reinforcement-learning framework, AutoCkt, and successfully test it on a transimpedance amplifier, two-stage amplifier, low dropout voltage regulator, and two-stage operational amplifier with negative $g_m$ load (the results of which are summarized in Table 2.10). Compared to prior optimization approaches, AutoCkt is on average 40× more sample-efficient than a genetic algorithm. In addition, the algorithm reaches state-of-the-art run-time sample efficiency, requiring just 9-20 simulation steps to find a design that meets a given target specification. We conduct accuracy, practicability, and generalizability analyses on the agent's designs to verify that the results are valid from a circuits perspective as this is crucial for a machine learning tool to be adopted by circuit designers for real applications. Comparing the agent's LDO design with an expert-designed LDO circuit, the proposed RL algorithm obtains 6.45% higher unity-gain bandwidth and 4.54% higher phase margin for -1.27% lower power.

By leveraging transfer learning, AutoCkt considers layout parasitics, and is 9.6× more sample-efficient than the state-of-the-art. We show that using only a 1-core CPU, the algorithm is able to design 40 LVS-passing designs for two-stage OTA with negative $g_m$ load in under 3 days.

Table 2.10: Summary of circuit topology results presented in this chapter.

| Topology | Average RSC (Schm./Layout) | # Tunable Parameters | Degree of Automation | Parasitics? |
|---|---|---|---|---|
| Two-stage amplifier | 27 | 9 | High | No |
| Transimpedance amplifier | 15 | 6 | High | No |
| LDO | 9 | 24 | High | No |
| Two-stage amp. $g_m$ load | 20/1 | 11 | Med-High | Yes |

# Chapter 3

# Reinforcement Learning for Parasitic-Sensitive Circuit Topologies

In order for an automated sizing tool to be practical, it must: 1) return valid results for a large range of target specifications, 2) provide reasoning for any failures in meeting a specification, 3) consider both layout parasitics and corner variations for complete end-to-end design, and 4) be automated, allowing most of the design effort to fall on the tool. This chapter expands the results in Chapter 2 by further assessing the generalizability of AutoCkt to two parasitic-sensitive circuit topologies: a two-stage transimpedance amplifier with negative $g_m$ load and two-stage folded cascode with biasing. We demonstrate that:

- a new combined distribution deployment algorithm improves RSC significantly,
- decisions the RL agent makes are explainable and analyzable in the context of circuits,
- the framework successfully reaches unique, valid, and practical performances, doing so in state-of-the-art run-time, at most $38\times$ more efficient than prior work, and
- it averages just 4 parasitic simulation steps to achieve a target specification post-layout for the folded-cascode, and 1 simulation for the two-stage transimpedance amplifier.

## 3.1 Modified deployment algorithm

In Chapter 2, we presented a deployment-generalization algorithm that takes an agent trained with schematic simulations, and deploys in an environment that solely considers parasitic simulations using BAG (re-summarized in Algorithm 3). We now reformulate the deployment algorithm to require less layout simulations, as well as be more robust to significant performance degradation in certain circuit topologies. This new generalization algorithm is summarized in Figure 3.1.

Instead of directly placing the trained agent in an environment that consists solely of running parasitic simulations (Chapter 2 [19]), schematic and parasitic distributions are combined by taking steps from both environments. This not only leads to faster run-time,

**Algorithm 3:** Deployment Algorithm with Only Layout Parasitic Simulations from Chapter 2

```
1  $\mathbf{o}^* \leftarrow \mathbb{R}^M$
2  $env \leftarrow$ parasitic simulator
3  while $True$ do
4      $\mathbf{o}, \mathbf{p} \leftarrow$ step($\mathbf{o}$, $\mathbf{o}*$, $\mathbf{p}$, $env$)
5      $layout\_reward$ = calc_reward ($\mathbf{o}$, $\mathbf{o}*$)
6      if $layout\_reward \geq 0.0$ then
7          return $\mathbf{p}$, $\mathbf{o}$, (reached $\leftarrow$ True)
8      step_count += H
9      if $step\_count > H$ then
10         return $\mathbf{p}$, $\mathbf{o}$, (reached $\leftarrow$ False)
11 end
```

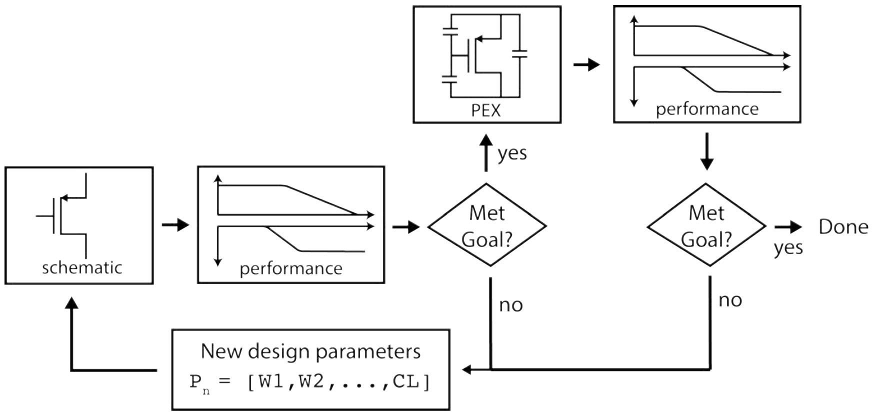


Figure 3.1: Parasitic deployment algorithm that shows how schematic and layout simulation distributions are combined to obtain a target specification.

as it avoids needless and costly parasitic simulations, it is also more robust to noise. This robustness is crucial, as the circuit topologies we select are highly sensitive to parasitics.

Upon receiving a target specification, the trained agent starts by running schematic simulations until it reaches $\mathbf{o}^*$. Instead of receiving a positive reward for that step for obtaining a valid solution in schematic, the same parameters the agent used to meet this target are simulated in an environment that considers layout parasitics. If the agent does not meet the target once parasitics are considered, the layout performance is given back to the agent, which then uses this information to adjust in schematic until it reaches another set of viable parameters which are used to run PEX. This process continues until the agent meets the target when parasitics are considered, or has reached the maximum allocated steps $H$, shown in Algorithm 4. This combination of schematic and layout simulation deployment allows the agent to traverse a distribution it knows, in order to better understand and generalize to a distribution it has not seen before, allowing it to be more successful in reaching a target.

**Algorithm 4:** Deployment Algorithm Considering Layout Parasitics

```
1  $\mathbf{o}^* \leftarrow \mathbb{R}^M$
2  $env_1 \leftarrow$ schematic simulator
3  $env_2 \leftarrow$ parasitic simulator
4  while $True$ do
5      $\mathbf{o}, \mathbf{p} \leftarrow$ step($\mathbf{o}$, $\mathbf{o}*$, $\mathbf{p}$, $env_1$)
6      $schematic\_reward$ = calc_reward ($\mathbf{o}$, $\mathbf{o}*$)
7      if $schematic\_reward \geq 0.0$ then
8          $\mathbf{o}, \mathbf{p} \leftarrow$ step($\mathbf{o}$, $\mathbf{o}*$, $\mathbf{p}$, $env_2$)
9          $parasitic\_reward$ = calc_reward ($\mathbf{o}$,$\mathbf{o}*$)
10         if $parasitic\_reward \geq 0.0$ then
11             return $\mathbf{p}$, $\mathbf{o}$, (reached $\leftarrow$ True)
12     if $step\_count > H$ then
13         return $\mathbf{p}$, $\mathbf{o}$, (reached $\leftarrow$ False)
14 end
```

## 3.2 Two-stage transimpedance amplifier with negative $g_m$ load

In Chapter 2, we demonstrated AutoCkt's successful functionality on a two-stage transimpedance amplifier with negative $g_m$ load with just an AC testbench [19]. We now extend the results on this circuit in TSMC 16nm technology to incorporate more target metrics, as well as reflect the modified training and deployment algorithm presented in Section 3.1.

Before doing so however, we discuss the manual equations and methodology used by experts to create this design.

### 3.2.1 Design equations and expert methodology

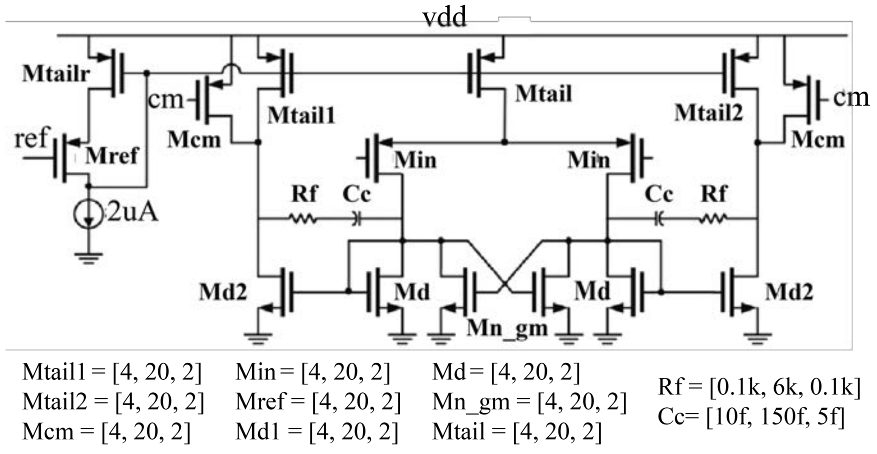


Figure 3.2: Two-stage transimpedance amplifier schematic.

The topology from Figure 3.2 is a standard miller compensated two-stage operational amplifier, with the first stage containing a negative $g_m$ load. The pMOS input stage has a positive feedback load, that creates a large differential small-signal resistance. The gain, $A_v$, is therefore:

$$A_v = g_{min} R_{out1} g_{md2} R_{out2} \tag{3.1}$$

where

$$R_{out1} = \frac{1}{(g_{ds,in} + g_{ds,d} + g_{ds,gm} + g_{md} - g_{mgm})} \tag{3.2}$$

and

$$R_{out2} = 1/(g_{ds,d2} + g_{ds,tail2}) \tag{3.3}$$

With miller compensation, the unity-gain bandwidth ($UGBW$) is determined by the transconductance of the input stage and the miller capacitor:

$$UGBW = g_{min}/C_c \tag{3.4}$$

The dominant pole, $f_{p1}$, can be estimated as:

$$f_{p1} = \frac{g_{min}}{(g_{min}R_{out1}g_{md2}R_{out2}C_c)} = \frac{1}{R_{out1}g_{md2}R_{out2}C_c} \tag{3.5}$$

The input transistor transconductance $g_{min}$ and $C_c$ are decided based on device characterization and bandwidth specifications.

The slew rate, $SR$, specification for an amplifier with miller-compensation is limited by the speed of the first stage charge miller capacitor $C_{c,p}$ and $C_{c,n}$:

$$SR = \frac{I_{tail}}{C_c} \tag{3.6}$$

This sets the lower bound of the current flow through the input pair, such that the size of these transistors can be determined. To achieve better settling and a large enough phase margin, the second pole, $f_{p2}$, can be adjusted and is determined by:

$$f_{p2} = \frac{1}{(g_{md2} * C_{load})} \approx (1-2) * UGBW \tag{3.7}$$

The second pole is located between $1-2\times$ the unity-gain bandwidth to ensure it is close to critical damping and will have a $45-60°$ phase margin.

Output swing ($V_{sw}$) is determined by the second stage:

$$V_{sw} = VDD - V_{ov,tail1} - V_{ov,d2} \tag{3.8}$$

Noise of the two-stage amplifier is mainly contributed by the first stage. The total noise current from the first stage is then:

$$in_{tot}^2 = [4kT(gamma_p * g_{min} + gamma_n * (g_{md} + g_{mgm})] * BW \tag{3.9}$$

where $BW$ is the bandwidth. The input-referred noise is therefore:

$$vn_{in}^2 = in_{tot}^2 / g_{min}^2 \tag{3.10}$$

### Expert design

An expert-designed iterative script was created in BAG for an AC-only testbench. The design process involves using the $g_m/I_D$ approach, created by sweeping a database of MOS parameters for varying current and terminal voltages and starts by checking whether the unity-gain bandwidth meets the given metric target. If not, the design parameters are set to some nominal minimum size transistor, which is used to simulate and get the resulting performance. If the design does not meet the unity-gain bandwidth target, the selected design specifications, along with their nominal current and voltage values are used to generate a new design. In the next iteration, the minimum size is increased for certain transistors, and the simulation steps continue until either the maximum number of steps is reached, or the design achieves the unity-gain bandwidth target. It is important to note that this design script does not consider other parameters (phase margin, etc.) and relies exclusively on iteration. Though the script successfully reaches some target specifications, it only does so within a narrow range of unity-gain bandwidths and must be verified by a designer to ensure that the designs are stable.

### 3.2.2 RL setup

Figure 3.2 shows the circuit topology and discrete tunable MOS, resistor, and capacitor parameters with their ranges [min, max, increment]. A load capacitance of $0.22pF$ is selected for training, which was taken from an expert design. The target metrics are:

- open-loop gain, $A_{vo}$
- phase margin, $pm$
- unity-gain frequency, $f_t$
- settling time, $t_{set}$
- dissipated power, $P$
- input-referred noise, $V_{noise}$
- differential output voltage swing, $V_{pp}$

The target specifications are selected from the range in Table 3.1, based on sampling regions around an expert-designed performance.

### 3.2.3 Training results

The minimum, mean, and maximum reward per trajectory for each iteration of training (represented as number of simulations or steps) is used to assess whether the algorithm trained successfully with schematic simulations. Figure 3.3 shows that the mean reward reaches zero after training has completed, meaning that the agent, on average, has learned to reach many training target specifications.

Table 3.1: Performance range sampled during training for full operational transimpedance amplifier.

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

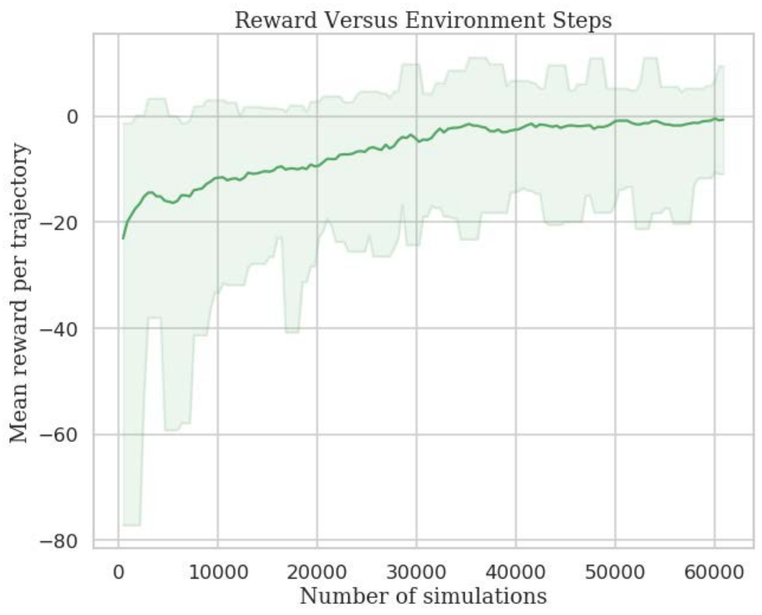


Figure 3.3: Mean episode reward over number of simulations for full two-stage transimpedance amplifier.

Table 3.2: Obtained schematic performances of trained agent for full operational transimpedance amplifier.

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

Table 3.3: Layout performance obtained by agent compared to expert for one design.

| Metric | Expert | Agent |
|---|---|---|
| Gain ($dB$) | 75 | 76.6 |
| Unity-gain frequency ($GHz$) | 0.16 | 0.17 |
| Phase margin ($^{\circ}$) | 69 | 65 |
| Settling time ($ns$) | 10 | 9.7 |
| Input-referred noise ($\mu V_{rms}$) | 1400 | 1358 |
| Differential voltage swing ($V$) | 0.57 | 0.59 |
| Dissipated power ($\mu$ A) | 0.54 | 0.52 |

Table 3.4: Rounded average required simulation count (RSC) for transimpedance two-stage amplifier in schematic and layout for 60 performances.

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

We test how well the algorithm understands the design space by giving the trained agent new and previously unseen target specifications. Table 3.2 shows the agent obtained 393 unique points with a wide range of schematic performances in each metric. In addition, the agent on average reaches these performances in 20 simulation steps.

The trained agent is deployed with layout and schematic simulations using the algorithm from Section 3.1. We compare the results to an expert design in Table 3.3 and show that the algorithm is able to out-perform the expert on every metric except phase margin, where there is a 5% degradation. In addition, this performance is obtained using 1 layout and 17 schematic simulations. This required simulation count (RSC) is summarized in Table 3.4 for the 60 unique tested layout performances the agent obtained, along with the results of varying random seed during training.

Compared to the prior deployment algorithm [19] which required 23 layout simulations to converge, this proposed approach uses 22.3× fewer run-time simulations. We note that this topology has significant degradation in schematic versus layout simulation results, with up to 28% difference between the metrics. The difference between schematic and PEX simulations for each metric, however, is mostly linear, meaning that the algorithm (and a potential layout parasitics model) could predict this degradation. In the next section, we introduce a more parasitic-sensitive topology to demonstrate the algorithm's robustness in cases where schematic and layout simulation results are significantly nonlinear and more difficult to model.

## 3.3 Folded-cascode amplifier

Analog-to-Digital converters (ADCs) are an important part of ASIC design, as they are responsible for translating real-world analog signals such as temperature or distance to a digital representation that can then be used for further processing [60]. An ADC is responsible for two processes that establish performance limits: quantization of the input signal which determines the resolution and error of the converter, and sampling the continuous input at certain uniform time intervals.

Several ADC architectures have been proposed to optimize quantization and sampling. Flash architectures, for example, are fast, needing just one ADC clock cycle to convert a signal, but require many comparator blocks to do so. Pipeline ADC designs achieve a higher resolution and dynamic range while handling wideband inputs, but with higher latency and potentially lower accuracy. Sigma-delta converters have even more latency, but yield the highest precision for low-bandwidth applications. While SAR ADCs have no pipeline delay, their accuracy depends heavily on comparator noise and Digital-to-Analog (DAC) linearity. With these different architectures, the design effort needed to complete an ADC system design is high, requiring an analog designer to not only understand the many tradeoffs, but also design and size each sub-block to meet a target specification known at the system level.

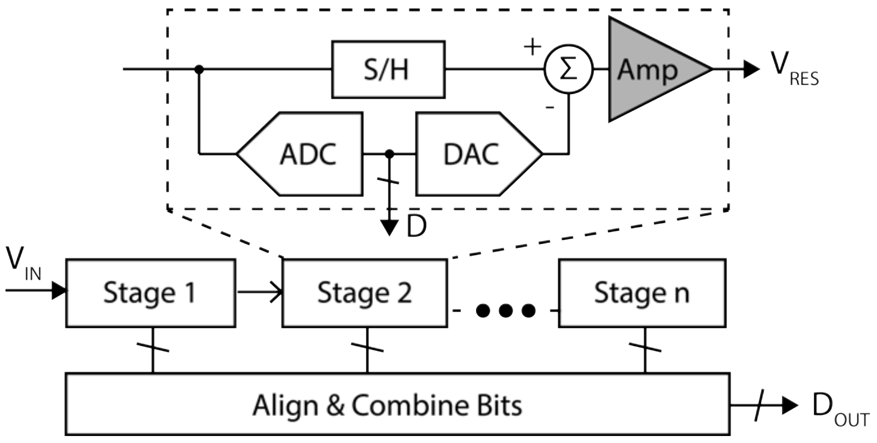
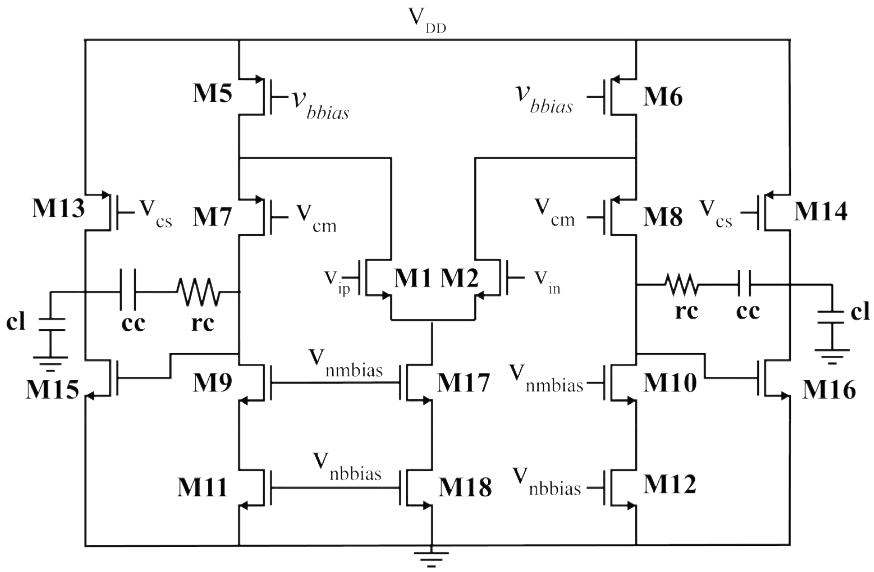


Figure 3.4: Pipeline ADC block diagram with amplifier circuit in gray.

We select a complete fully differential two-stage folded-cascode amplifier with biasing circuitry to evaluate AutoCkt's performance on a more complex, parasitic-sensitive topology, adapting the folded-cascode from [61] to enable functionality in TSMC 16nm technology. The target specifications are selected to match practical requirements in a pipeline analog-to-digital converter (ADC), whose block diagram is shown in Figure 3.4.

The folded-cascode consists of biasing and amplifier stages, with ideal common-mode feedback, illustrated in Figures 3.5 and 3.6. In total, the circuit has 42 parameters, including transistor sizes, load and compensation capacitors and resistors, and a bias current source.

This circuit not only has a complex design space, but also shows significant behavioral changes once layout parasitics are incorporated, making it an ideal candidate to test the robustness of this reinforcement learning methodology.



cl = [8f, 30f, 2f] or [0.1p, 10p, 1p]
cc = [2f, 24f, 2f]
rc = [0.6k, 1.7k, 0.1k]
M15 = M16 = [2, 24,2]
M5 = M6 = [2, 24,2]
M2 = M8 = [2, 24,2]
M9 = M10 = [2, 24, 2]
M11 = M12 = [2, 24, 2]
M13 = M14 = [2, 24,2]
M17 = [2, 24, 2]
M1 = M2 = [2, 24, 2]
M7 = M8 = [2, 24, 2]

Figure 3.5: Folded-cascode core topology: 22 parameters total.

### 3.3.1 Schematic training setup

To obtain a performance baseline in $16nm$ technology, the desired target specification ranges are determined. To understand the practicality and feasibility of this chosen target specification, first and second-order approximation are used with the following steps:

- Assuming a conservative nominal overdrive voltage $V_{ov}$, defined as the threshold voltage subtracted from the device gate-to-source voltage [29] as $100mV$ for this technology, the nMOS output stage of the class A amplifier can swing between $100mV$ and $900mV$. Thus, the nominal swing is $800mV$.

- To determine a desired settling time, the sampling rate for the ADC is selected first. In a pipeline ADC design, the sampling rate is decided solely by the delay of a single

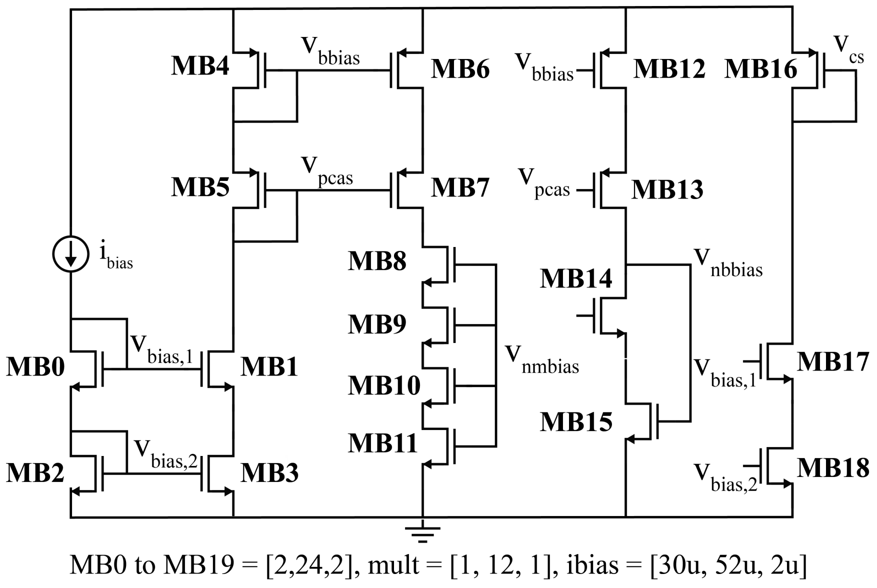


Figure 3.6: Folded-cascode biasing topology: 20 parameters total.

stage. For larger input swings, settling time will be comprised of both nonlinear slewing and recovery and linear settling of the amplifier. Assuming half of a clock cycle is used to settle linearly, the sampling rate, $f_s$, is $1/(2 * t_{set})$ where $t_{set}$ is the settling time, assuming an ideal clock with $50-50$ duty cycle. If a nominal $250MHz$ sampling rate is targeted, the desired settling time would then be $2ns$. Practically, however, some part of the settling time is allocated to generate a non-overlapping clock. Therefore the design should aim to be at least 15% below this value, $1.7ns$.

- To determine a feasible bandwidth constraint, we analyze the transfer function of the folded-cascode topology:

$$H(s) = \frac{1}{\frac{s^2}{w_u w_2} + \frac{s}{1/w_u + 1/w_2} + 1}$$

With higher-order responses, it is important to consider damping, as this strongly affects settling time. In particular, the fastest settling time occurs when the circuit is critically damped, meaning that $w_2 = w_u$ and the phase margin is 45°. In practice however, phase-margin requirements must be at least 60° to prevent oscillatory behavior. Assuming that $r_c$ is chosen such that the circuit is ideally compensated, $w_2 = 1.8 * w_u$, a linear time invariant system $H(s)$ in unit feedback can be formulated, and the resulting step response can be measured over time. The error for this response is calculated by $1 - s$ where $s$ is the output step response. ENOB settling error is then calculated:

$$err = 10 * log(\frac{1}{2^N}) \tag{3.11}$$

for 8 and 10-bits which is $-24.1dB$ and $-30.1dB$, respectively. Finding the intersection between Equation 3.11 and the output error response results in settling time that can then be converted to a targeted gain bandwidth. Nominally tolerance for error is added in this time component by reducing it to one-half of its original value. Thus the bandwidth for each application is $980MHz$ for 8-bit and $1.20GHz$ for 10-bit.

- To determine a gain constraint, technology limitations and constraints are determined by a pipeline ADC. The maximum $g_m$ and $r_o$ is calculated for a 16nm process bounded by 20 fingers and $25\mu A$ of drain current, for a fixed minimum channel length $16nm$. These values are used to approximate the highest theoretical gain:

$$A_V = gm_1(R_{d7}||R_{d9}) * gm_{15}(r_{d13}||r_{d15}) \tag{3.12}$$

where

$$R_{d7} = r_{d7}[1 + gm_7(r_{d1}||r_{d5})] \tag{3.13}$$

and

$$R_{d9} = r_{d9}(1 + gm_9 r_{d11}) \tag{3.14}$$

This results in the maximum gain being $120dB$. This value is in practice much lower due to parasitics. To obtain a more reasonable target, total error ($err_{tot}$) is used:

$$err_{tot} = \frac{1}{2^N} \tag{3.15}$$

The gain can be calculated for 8-bit and 10-bit ADC using this error with:

$$20 * log(\frac{1}{2^N}) \tag{3.16}$$

The associated values are: $48dB$ and $60dB$, respectively. In practice, the gain should be higher than this minimum to account for margin.

- In a pipeline ADC, thermal noise is the sum of all of the input-referred noise values from each pipeline stage. The noise contribution from later stages, however, diminishes because it is attenuated by the gain stages that precede them. Because of this, the noise budget is partitioned such that the first sample-and-hold stage contributes approximately half of the noise, the first stage has one fourth of the total input-referred noise, and all other stages must be designed within the rest of the noise budget. In an N-bit ADC, total input-referred noise is equal to the quantization plus thermal noise:

$$N_{thermal} + N_{quant} = \frac{LSB^2}{12}$$

Assuming a $1V_{pp}$ differential input signal, the total input-referred noise for an 8-bit ADC is $1.1mV_{rms}^2$, and the amplifier maximum noise budget would then be $790\mu V_{rms}$. In practice, realistic designs have some tolerance built in. For this design, a 5% tolerance is added, thus adjusting the noise budget to $750\mu V_{rms}$. For the corresponding 10-bit ADC, the input-referred noise is then $560\mu V_{rms}^2$, making the noise budget for the first stage $380\mu V_{rms}$. In general, however, these noise constraints are empirical, as there is no direct connection between the noise budget in the ADC stage and the amplifier noise simulation results. Without explicitly knowing this target value, placing a hard constraint on noise is inaccurate and not representative of the application. Because of this, the constraint placed in the above analyses is loosely defined and 40% variability is added to account for the fact that noise can be reduced by upscaling the ADC capacitors or the entire amplifier to provide enough drive current.

- The noise limitation also places a constraint on the load capacitance ($C_L$), and $kT/C$ dictates that the range of load capacitances must be in pico-Farads. This in turn influences the AC behavior of the circuit, now making the output pole dominant. In the results, we discuss two cases: one where the $C_L$ given to the agent ranges from $0.1 - 10pF$, supporting established equation-based analyses, and the other where $C_L$ varies from $8 - 30fF$. The second case gives important insight into how this agent functions in other non-traditional design cases.

The constraints are summarized in Table 3.5. Note that this analysis is conducted to show that the target specifications are in the range of interest for a practical design application, and is within the bounds of feasibility when hand-designing the circuit, for the purposes of a baseline. This algorithm bears no requirement to having this information.

Table 3.5: Derived target specifications for folded-cascode in pipeline ADC application.

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

Table 3.6: Parameter constraint selection.

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

The tunable parameters for this circuit topology include length and number of fingers for each MOS device, feedback resistance, feedback and load capacitances, multiplier, and the input current source. Each parameter range is derived from the BAG layout generator used to consider layout parasitics, and has 11 unique values to select from. We focus on uniform channel length devices when parasitics are included, limited by the simplicity of the BAG layout generator.

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

Thus, in the experiments we provide three different parameter versions, one where the agent receives no outside constraint assistance, one where the agent receives basic necessary constraints, such as matching, and the last where the agent receives matching and easily coded yet still limited constraints, such as certain biasing circuitry transistors having widths and lengths multiples of one another. These three versions are summarized in Table 3.6. We note that no test involves explicitly formulating design equations, as that prevents the tool from being automated.

### 3.3.2 Design equations and expert methodology

Before discussing results, we first analyze the design equations experts use to manually size the folded-cascode in the case where the load capacitance creates a dominant pole in the system. This allows us to assess the RL agent's efficacy by comparing and understanding the similarities between both algorithms.

Based on a slew rate, $SR$, requirement, the bias current through the load capacitor when a large differential input signal is applied, $I$, can be calculated for a certain $C_L$:

$$SR = \frac{I}{C_L} \tag{3.17}$$

The slew rate determines the lower bound of the current flow through the input pair, meaning that their size can be decided. To decide the size of the other transistors in the first stage, the current of the branches needs to be determined. The current flow through the input pair should be larger than or equal to the current of the cascode branches to lower the power consumption and maximize gain. Initially, in the sizing process, the currents can be made equal. Transistors $M_{5,6}$, $M_{11,12}$ and $M_4$ are sized to provide current for these branches. The ratio between $M_{11,12}$ and $M_4$ should be the ratio of current between them.

The gain-bandwidth ($GBW$) is:

$$GBW = \frac{g_{m1}}{C_L} \tag{3.18}$$

where $gm_1$ is the transconductance of $M_1$. The gain, $A_V$, is determined by multiplying the gain from the first ($A_{V1}$) and second stages ($A_{V2}$) together:

$$A_{V1} = g_{m1}(R_{d7}||R_{d9}) \tag{3.19}$$

where

$$R_{d7} = r_{d7}[1 + g_{m7}(r_{d1}||r_{d5})] \tag{3.20}$$

and

$$R_{d9} = r_{d9}(1 + g_{m9}r_{d11}) \tag{3.21}$$

The second stage gain is:

$$A_{v2} = g_{m15}(r_{d13}||r_{d15}) \tag{3.22}$$

$$A_V = A_{V1}A_{V2} \tag{3.23}$$

The output impedance of the first stage is:

$$Rout_1 = R_p||R_n \tag{3.24}$$

where

$$R_p = r_{o9}[1 + g_{m9}r_{d11}] \tag{3.25}$$

and

$$R_n = r_{o9}[1 + g_{m7}(r_{o1}||r_{o5})]. \tag{3.26}$$

Transistors $M_{5,6}$ and $M_{11,12}$ are responsible for providing high output resistance, which means that these devices must have a proper drain-to-source voltage to improve output impedance. All transistors in the cascode branch, $M_{5-12}$, require some margin between their drain-to-source voltage and the overdrive voltage in order to guarantee small-signal impedance.

The voltage swing, $V_{sw}$ is determined by the output stage:

$$V_{sw} = VDD = V_{ov,14} - V_{ov,16} \tag{3.27}$$

To sizes of the output devices are dependent on both the output voltage swing, and driving capability. These devices must provide sufficient current when the output voltage reaches its required voltage swing. The ratio between $M_{14}$ and $M_{16}$ is adjusted to provide enough gain, and to ensure the input common-mode voltage is reasonable for the output of the first stage. Initially, these transistors are sized to be a unit output stage, and then scaled up iteratively to meet the requirements of the capacitive load.

The biasing transistor sizes are determined by the currents required in the cascoded and tail branches of the main amplifier. $MB_{1,3}$, $MB_{2,4}$ and $MB_{19,20}$ mirror the reference current $i_{bias}$ to different branches. Their are not critical but need to be large enough so that there is enough headroom for the pMOS devices in the folded-cascode. $MB_{19}$ and $MB_{14}$ provide the require dbiasing for $M_{9,10}$, $MB_{15}$ provides the biasing for $M_{11,12}$. Both of their sizes should be the desired current ratio. The ratio between $MB_{16,17,18}$ and $M_{13,14}$ needs to be the same as the current ratio between the bias branch and the output stage. $MB_{5-8}$ provide the bias voltage to $M_{5,6}$.

#### Expert methodology

Folded-cascode design methodologies generally encompass a combination of heuristics and circuits intuition. For example, authors in [61] use a $g_m I_D$ approach, and begin by sizing the transistors based on nominal current requirements in each branch of the folded-cascode, which are then looked-up and iterated on until a specification is reached. For those target metrics that are not explicitly derived in equation-analysis, like settling time, prior work tends to approximate them later in the design process, in system-level simulations, or incorporate it into the sizing process only when AC metrics have been satisfied [62].

### 3.3.3 Schematic training and deployment results for folded-cascode

In this section, we show the results of training and deploying the agent on the two-stage folded-cascode, as well as additional analyses to understand the effect of varying targets, constraints and performance. We consider two cases, 1) where the load capacitance is in pico-Farads, following conventional sizing methodology, and 2) the load capacitance is in femto-Farads, creating less practical, but more unique design cases. In addition, we compare the results to prior work to show that this algorithm is up to 38× faster in converging to a target during deployment.

### Higher load capacitance results

We will first discuss the results of training the folded-cascode with higher load capacitance in order to compare against conventional sizing methodology. This is important as it validates the designs produced by an agent. The means for doing this comparison, however are not straightforward. Because traditional design methodology, especially for the folded-cascode, relies heavily on iteration, the equations do not necessarily correspond to the exact sizes that produce a feasible design that meet a target specification. What we can do, however, is assess the similarities in performance and efficiency to determine validity.

To do so, the trained agent and the expert are given a target specification to meet, which, in this case, designs for a lower-gain higher bandwidth application. The obtained performances for these designs is shown in Table 3.7. We also calculate the efficiency of an agent-produced and expert design using $f_t C_L / i_{bias}$. The table shows that the agent achieves a better efficiency metric, validating that the designs are comparable.

To understand the practicality of the agent-produced design, we analyze the noise, $V_n$, caused by the dominant load capacitance using:

$$V_n = kT/C_L \tag{3.28}$$

where $k$ is Boltzmann's constant, $T$ is the temperature in Kelvin, and $C_L$ is the load capacitance. With an $1pF$ load capacitance, the $V_n$ is therefore $64\mu V$. Assuming that the ADC requires a $4\times$ higher noise budget, the LSB for the ADC is $256\mu V$. This therefore gives an input constraint in designing the ADC and establishes that the design produced by the agent is feasible.

Table 3.7: Comparison of efficiency between agent-obtained and expert-designed performances.

| | Expert design | Agent design |
|---|---|---|
| UGBW ($GHz$) | 10.0 | 10.5 |
| Gain ($dB$) | 66.1 | 49.4 |
| Noise ($\mu V_{rms}$) | 352 | 352 |
| Power ($mA$) | 3.33 | 1.24 |
| PM ($^\circ$) | 85.2 | 66.3 |
| Tset ($ns$) | 1.7 | 0.89 |
| Vswing ($V_{pp}$) | 0.79 | 0.83 |
| $C_L$ ($pF$) | 1 | 1 |
| Efficiency ($MHz * pF/mA$) | 3000 | 8500 |

### Varying target specifications during training

We now move to discussing results with the lower $C_L$ capacitance. The mean episode reward per number of steps is depicted in Figure 3.7, and shows that the agent successfully reaches the stopping condition by obtaining many target specifications.

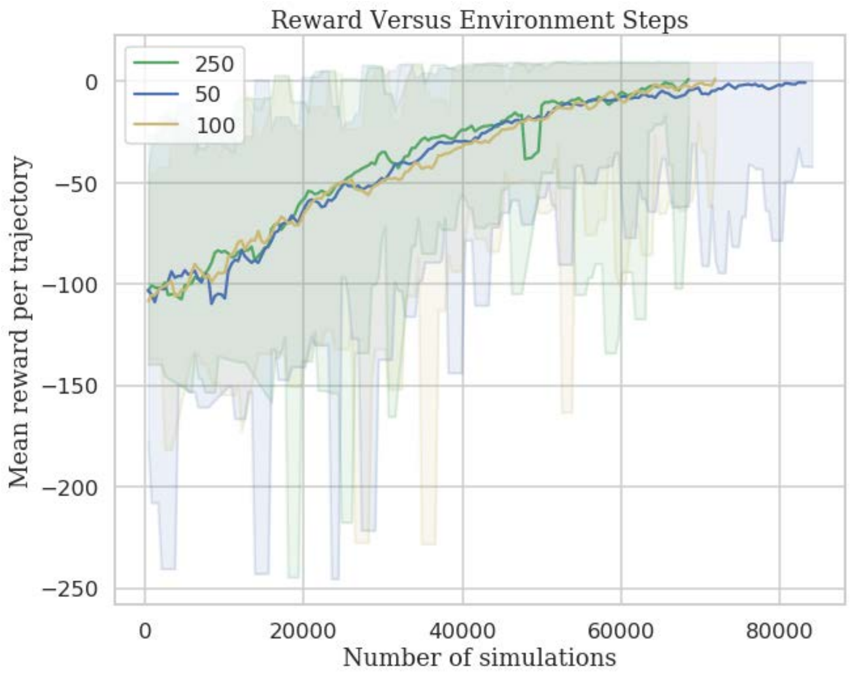


Figure 3.7: Mean reward per trajectory for 50, 100, and 250 training target specifications.

Figure 3.7 also shows the effect of varying the total number of training target specifications given to the agent to meet. It shows that the number of simulations required to train is inversely correlated to the number of targets given to the agent, with more targets helping the agent train faster. This, however, is not significantly different compared to the total number of simulations, with all three experiments falling within 12000 simulation steps of each other. These results are expected, as the agent is able to randomly select targets for each trajectory at every training iteration. With a higher number of targets, the agent has a larger variation with this selection, meaning that a particular sample of targets could help the agent reach the stopping condition faster. We do note that there appears to be a larger step-to-step variation due to this.

To understand intuitively what Figure 3.7 means in the context of the circuit sizing problem, the number of target specifications met by the agent during training is calculated. The results are summarized in Figure 3.8, which also shows the effect of varying the number of training target specifications. The agent on average meets just over half of the targets across the three different experiments. As we will show in the failure analyses in Section 3.3.4, this 55% success rate does not mean the agent fails at learning the relationship between parameters and performance for this topology. Instead, we demonstrate the targets that have been chosen are pushing performance boundaries for this parameter range and technology.

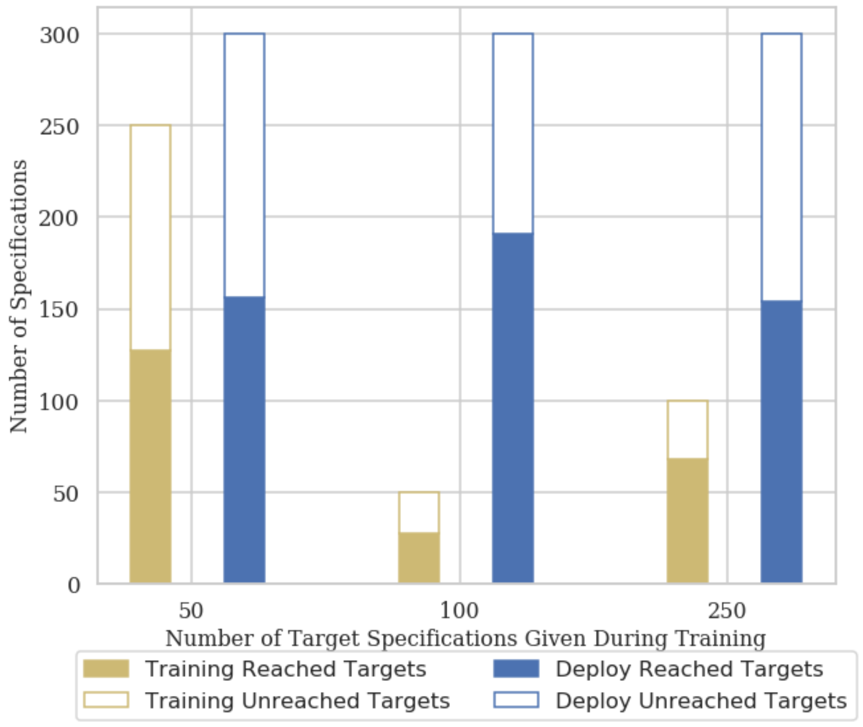


Figure 3.8: Deploying trained agent on target specifications not seen before: folded-cascode.

### Varying number of constraints

Figure 3.9 shows the results of varying the number of constraints applied to the agent during training. This plot shows that as the number of parameters increases, the longer it takes the agent to train. In all cases, the agent is able to successfully maintain an upwards increase in mean reward and reach the stopping condition. The no constraints model, however, requires considerably more simulation steps to converge. This is likely due to difficulty in obtaining feasible design regions when the parameter count is very high. In our studies, we have found that randomly selecting parameters in the space has an 85% chance of not passing DC simulation constraints, which could explain the initial flat slope for this experiment.

### Deployment with schematic-only simulator

The trained agent is deployed on 300 new target specifications it has never seen before, in the same range as the randomly sampled targets during training in schematic only, and summarize the results in Figure 3.8. The percentage of reached versus unreached targets is similar to the results of training, with approximately 50% of them being successful. This in turn means that the agent does not overfit on the targets given to it during training.

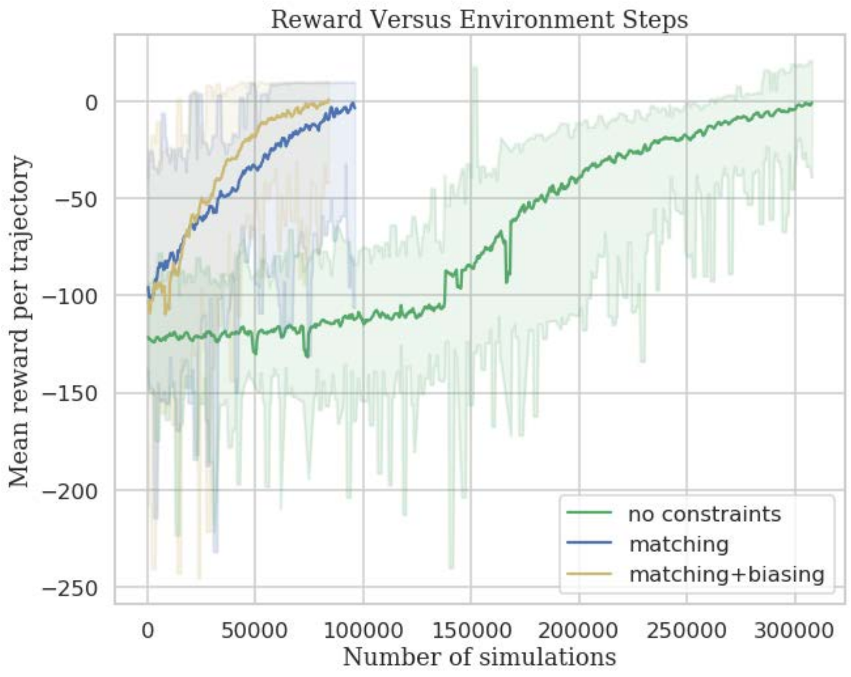


Figure 3.9: Mean reward per trajectory for varying number of constraints: folded-cascode.

In addition, the trained agent requires on average just 12.5 simulation steps to converge to a set of parameters that meets a given target specification. We compare this RSC with prior state-of-the-art, which include genetic algorithms and their variants, in Table 3.8. To generate the best possible comparison, the hyperparameters for each of these approaches is optimally selected based on the fastest convergence on a test target specification. These hyperparameters are then fixed across the rest of the target specifications. We note that for the genetic algorithm boosted with neural network [16], we do not tune the neural network's hyperparameters, as these parameters are not modified in this reinforcement learning framework. Table 3.8 also includes the RSC to train the RL agent, which is considerably higher than prior work. This training cost is amortized, however, in two ways: 1) reusing the trained agent on other target specifications for the same circuit topology leads to a run-time convergence that is at least 38× faster than prior work, and 2) deploying the trained agent on layout simulations reduces post-layout run-time convergence, as we will show in Section 3.3.5.

Table 3.8: Comparison of prior analog sizing methods with required simulation count (RSC) to converge and number of specifications reached versus tested.

| Prior Work | RSC | # Specs Reached / # Specs Tested |
|---|---|---|
| GA + NN [16] | 494 | 4/50 |
| GA [63] | 51973 | 1/1* |
| AutoCkt (training) | 71001 | N/A |
| AutoCkt (deploy) | 13 | 191/300 |

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

### 3.3.4 Reachability analyses in schematic trained agent for folded-cascode

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

#### Variability and validity of reached performances and parameters

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

The minimum, maximum (at most 11), and mean array index values for each tunable parameter in the folded-cascode are shown at each end of the gray line in Figure 3.10, as well as the black circle, respectively, for cases where the agent successfully reaches a target specification. The variability in parameter values clearly shows that the agent understands and reaches performances with diverse exploration over the range of parameters.

The minimum, mean and maximum obtained performances are summarized in Table 3.9, showing that the agent reaches a diverse range of performances.

To analyze the validity of the agent's designs from a circuit's perspective, we examine transistor operating points to ensure that devices operate in the correct regime. For this topology, the overdrive voltage values for each of the obtained performances are calculated and averaged. The results are summarized in Figure 3.11 and show that the framework finds reasonable and valid operating regions for each amplifier transistor.

#### Analysis on unreached target specifications

The distribution of unreached target specifications is analyzed according to the performances the agent converges to. In such cases, we look at the number of target metrics met, and summarize the results in Table 3.10. This table shows that 89.7% of the unreached targets

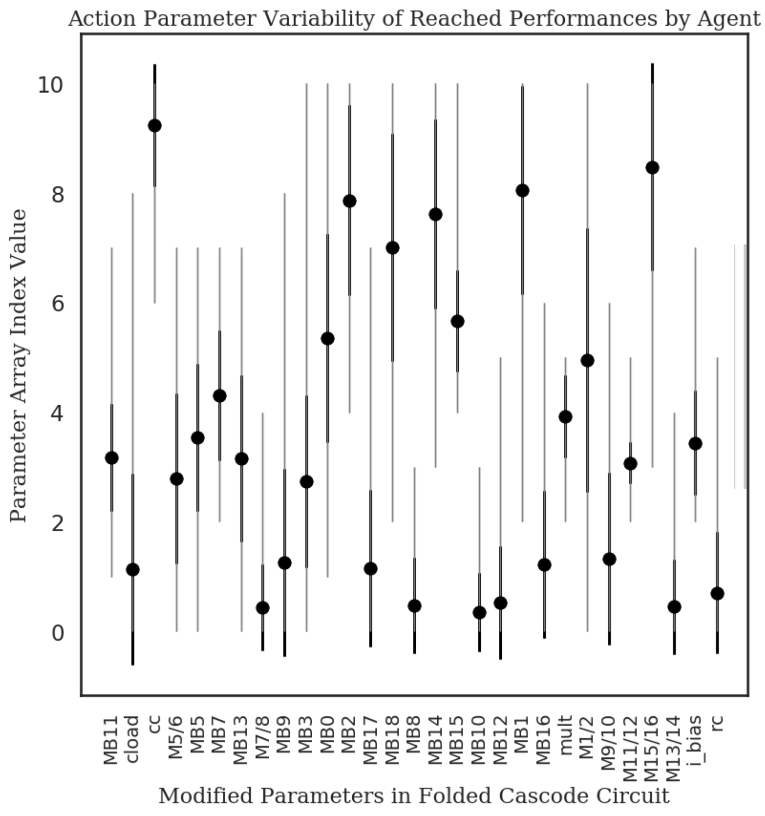


Figure 3.10: Parameter variation for reached performances: folded-cascode.

Table 3.9: Reached performance ranges by agent ($100o^*$).

| Metric | Minimum | Mean | Maximum |
|---|---|---|---|
| UGBW ($GHz$) | 2.67 | 7.84 | 14.8 |
| Gain ($dB$) | 56.7 | 63.2 | 65.7 |
| Noise ($\mu V_{rms}$) | 303.4 | 380.4 | 531.1 |
| Power ($\mu A$) | 370.1 | 613.0 | 803.1 |
| PM ($^\circ$) | 59.6 | 73.4 | 93.7 |
| Tset ($ns$) | 1.2 | 3.6 | 10.5 |
| Vswing ($V$) | 0.62 | 0.86 | 0.97 |

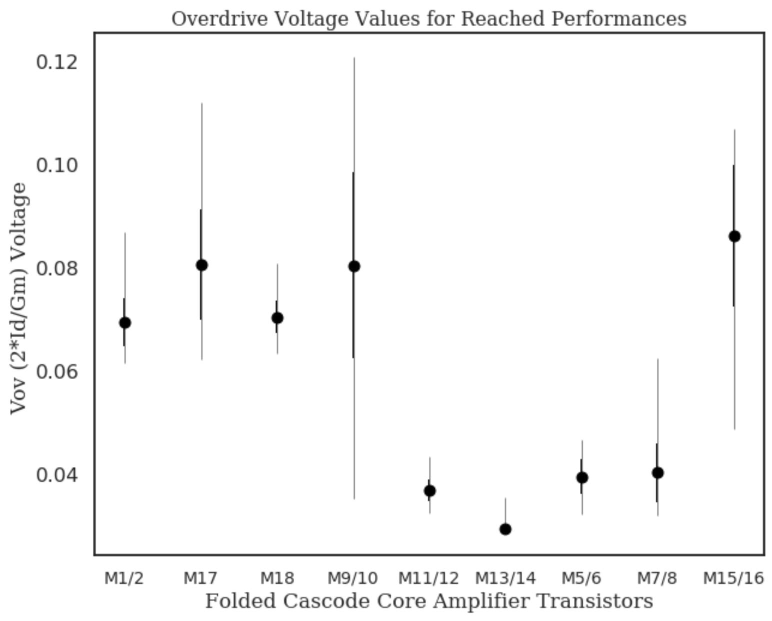


Figure 3.11: Overdrive voltage ($V_{ov}$) analysis for obtained performances: folded-cascode.

Table 3.10: Reached performance metric distribution for unreached targets.

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

consist of greater than five metrics being met, out of a total of seven. This means that the agent optimizes and obtains performances that are close to the target. Figure 3.12 further shows the unmet target specifications, categorized by what percent of metrics are not met. From this plot, it is clear that settling time, phase margin and gain constraints are a significant portion of the unmet distribution (power is ignored as it is not a hard constraint). The percentage error between the obtained metric performance and its associated target is also shown in the figure. We note large errors in unity-gain frequency, settling time, and voltage swing, but posit that this is due to the selected targets pushing the performance boundary for this circuit topology, as we will analytically show in the next sub-section.

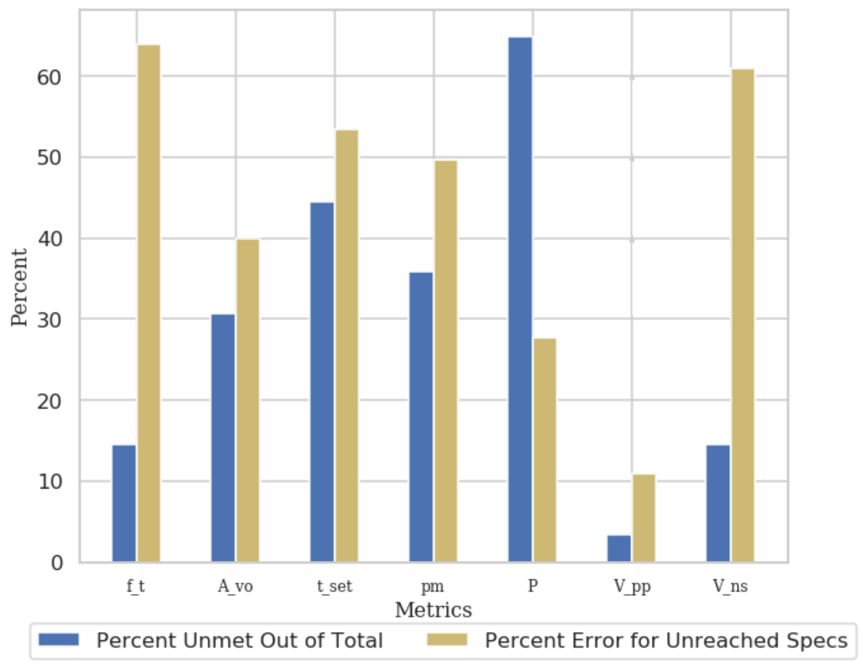


Figure 3.12: Metric failure percentage and percent difference to target for those failed metrics, where $f\_t$ is unity-gain frequency, $A\_vo$ is gain, $t\_set$ is settling time, $pm$ is phase margin, $P$ is power, $V\_pp$ is voltage swing, and $V\_ns$ is noise.

## Distribution of unreached target specifications

We analyze the distribution of unreached targets, and group each metric based on whether they are within 15% of the maximal (for metrics being maximized) or minimal range (for metrics being minimized) initially used to generate the targets during training, with the exception of phase margin which only has a 5% range to account for the smaller variation. The results are summarized in Figure 3.13. This histogram shows that 98.6% of distribution of target specifications the agent does not meet stems from at least one metric nearing its upper (maximized metrics) or lower bound (minimized metrics). In addition, Figure 3.13 also plots the unmet target metrics over the total number of targets classified within each bound, which increases with more metrics being in the toughest bound. Because these target metrics are randomly sampled from a fixed range, the probability of sampling six or seven metrics within the toughest bound is very low, explaining why fewer targets exist in these categories.

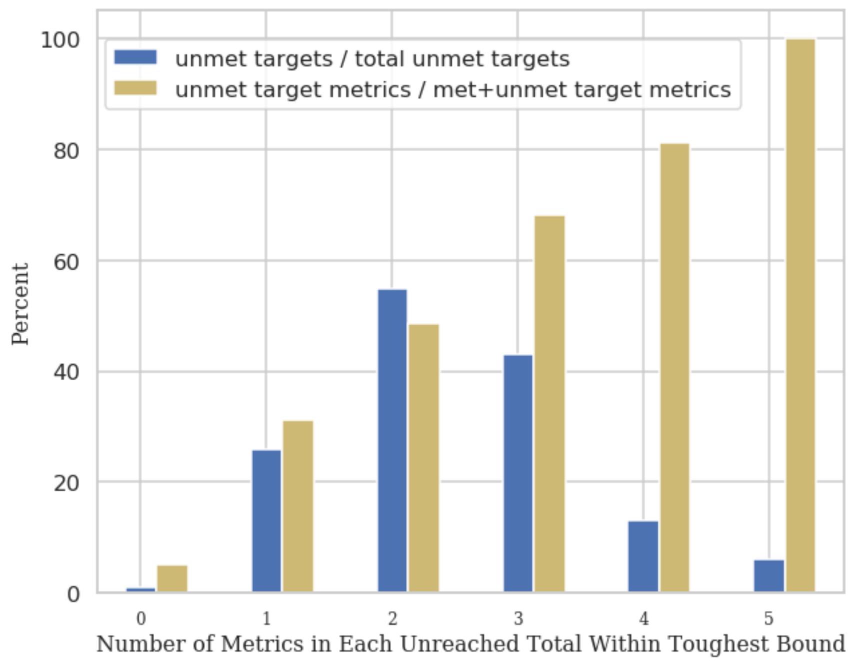


Figure 3.13: Distribution of unreached targets binned by number of metrics that are within their toughest bound.

Figure 3.14 plots a heatmap of each of the metric tradeoffs that contribute to the histogram from Figure 3.13. The highest number of tradeoffs occurs between output voltage swing and the other metrics, including power and phase margin. From a circuits perspective, this is intuitive as voltage swing determines the bounds of other metrics. In addition, gain and power, gain and swing, and noise and power are featured as well. It is apparent that these common circuit tradeoffs are being considered by the agent during the training process.

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

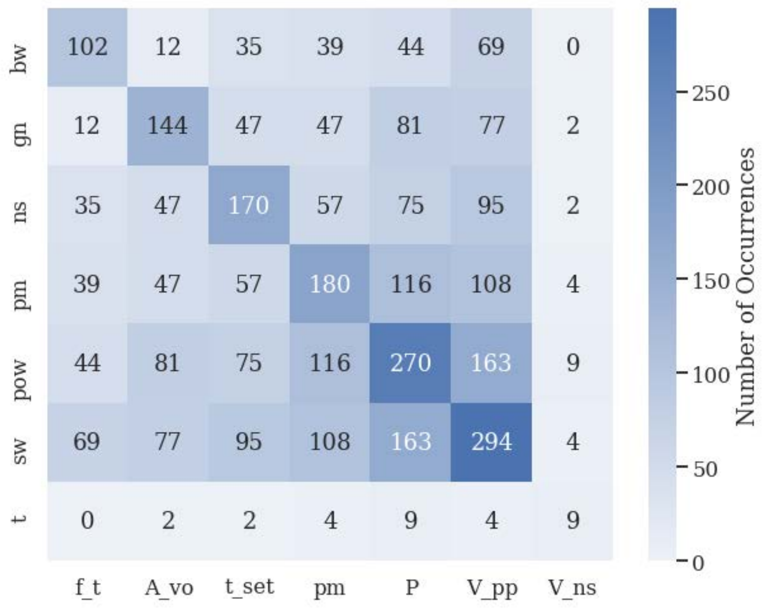


Figure 3.14: Heatmap of number of occurrences of different metric tradeoffs in unreached target specifications, where $f\_t$ is unity-gain frequency, $A\_vo$ is gain, $t\_set$ is settling time, $pm$ is phase margin, $P$ is power, $V\_pp$ is voltage swing, and $V\_ns$ is noise.

### 3.3.5 Generalization to layout parasitics

Prior analog sizing tools either lack the ability to consider post-layout extracted (PEX) simulations due to RSC inefficiency and lack of automated generation of layout, or consider them via approximated parasitic models [41, 42, 43], which can prove inaccurate in cases where there is significant unpredictability and performance degradation between schematic and layout simulations across a wide sizing range. In order to make the design of analog sizing practical, sizing frameworks must consider layout and demonstrate its functionality on true circuit parasitics. In this section, we discuss how BAG is utilized to generate layouts for the folded-cascode, analyze why a combination of schematic and PEX simulations is the most successful in reaching a target specification, and show how the trained RL agent performs with layout parasitics.

#### Two-stage folded-cascode BAG generator

The two-stage folded-cascode, designed by Zhaokai Liu, is created by two sub-generators: the core amplifier and the biasing circuitry block. These generators are made to encompass a wide range of transistor sizes. The amplifier core assumes the transistors for each differential side are symmetrical and matched, and are grouped into two parts for ease of layout generation: the first consists of the tail current, input pair, and the p-type cascode transistors and the second contains the n-type and output stage transistors. This grouping allows consistent generation of LVS-passed layouts. Dummy transistors are added in empty regions to guarantee good matching.

The bias circuit generator from Figure 3.6 has three sub-blocks: the p-type cascode, n-type cascode, and tail-current biasing. These sub-blocks are connected vertically allowing for easier modification of sizes. In addition, they can be adjusted without needing a complicated routing algorithm. Both amplifier core and bias circuit have supply routing at two sides. Connections between these blocks use a higher metal layer.

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

### Combined Schematic and Layout Deployment Methodology

In order to understand why a combination of schematic and layout simulations is the most successful in reaching a given target specification, we first analyze the distributional difference between these datasets. Figure 3.15 shows schematic and PEX simulation results for the same set of parameters. These plots show that layout parasitics affect each metric differently, with some metrics, including phase margin, settling time, and gain having distinct non-linear patterns. In particular, only 10% of the points meet the phase-margin requirement of 60° in layout. Gain also contains a sharp dropoff, attributed to self-loading, that degrades performance significantly compared to its corresponding schematic design. These features affirm that layout effects are significant for this topology, and by using a simple linear or neural network model to predict parasitics allows for optimization in a very limited range, restricting the applicability to lower-performance, non-parasitic dominant circuits. Principal component analysis is conducted in Figure 3.15 to show that these datasets form separate, almost non-overlapping clusters in the design space, with parasitic results having less spread and a different centroid location. This reinforces that simply taking parameters for a given performance that satisfy a target specification in schematic and translating it to parasitics does not yield reliable results. Additionally, the sensitivity of every tunable parameter to each target metric is shown in Figure 3.16, by calculating the change in performance when a single parameter is modified. We find that the multiplier, nMOS load transistors, and certain bias transistors cause significant changes to the metric performances, further illustrating that self-loading is a significant factor in performance degradation in layout.

Two trajectories are plotted in Figure 3.17, showing where the deployed agent only runs layout simulations [19], as well as the proposed schematic and layout simulation approach. The results show that the layout simulation approach does not converge to a specification that meets the target, shown as the green horizontal line (scaled to 0.0), despite improving the reward over the course of the trajectory. This approach (in yellow), allows the agent to traverse the schematic simulation space, routinely checking layout-parasitic simulation

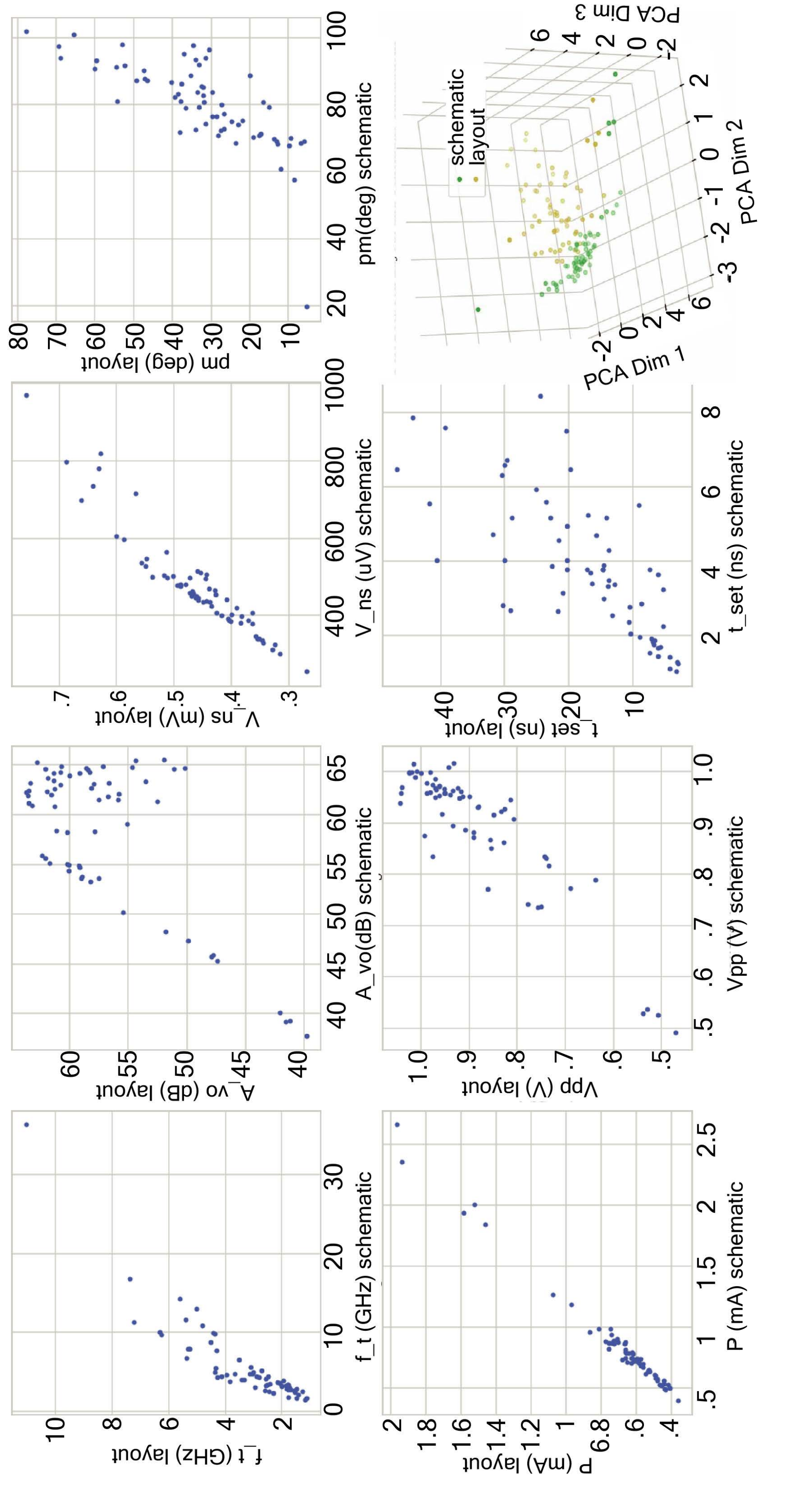


Figure 3.15: Scatterplot showing the difference between schematic and layout simulation results: folded-cascode circuit.

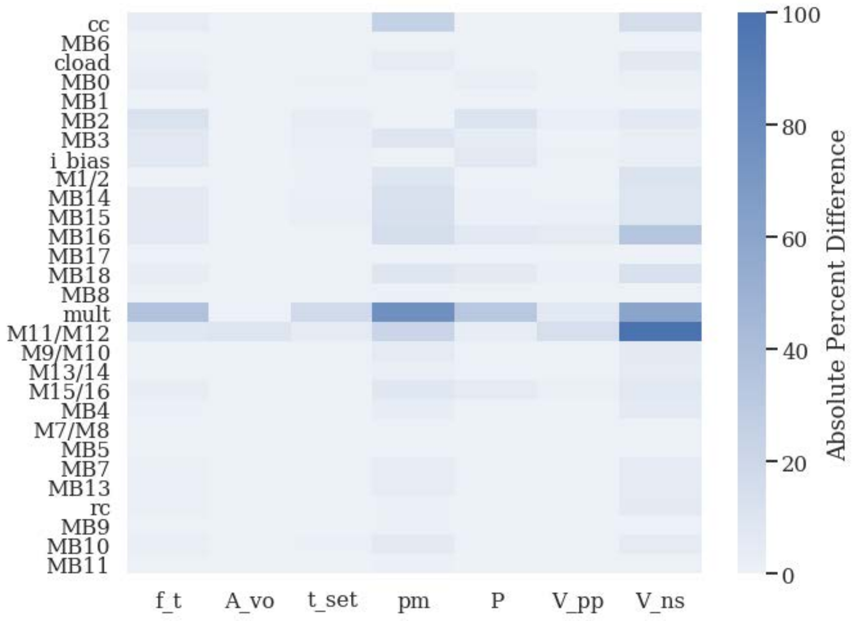


Figure 3.16: Sensitivity heatmap that shows the absolute percent change in layout metric performance when a selected circuit parameters are individually modified in the design: folded-cascode.

results and modifying the parameters in a design space that it better understands. If we had simply taken the first schematic simulation parameters that met the target and used this to run PEX, depicted as the theoretical schematic simulation in Figure 3.17, the target specification would not have been reached. This combination of schematic and layout simulation deployment allows the agent to traverse a design space it knows and understands, in order to generalize to a distribution that it has not seen before, allowing it to be more successful in reaching a target [64].

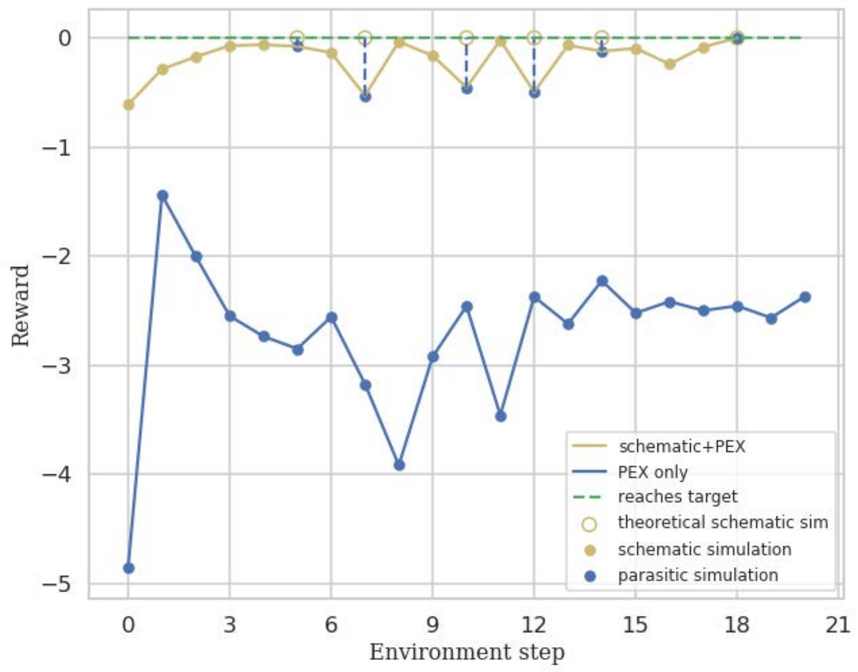


Figure 3.17: Trajectory results when agent is run with a combination of schematic and layout simulations, versus only layout simulations.

## AutoCkt with layout parasitics

Table 3.11: Reached performance ranges with layout parasitics.

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

We test the combined schematic and layout deployment methodology by giving the agent a range of target design specifications to reach. Table 3.11 shows the minimum, maximum, and mean obtained layout performances on 60 trajectories. We find that the framework successfully reaches a range of valid PEX performances, meaning that it successfully sizes parameters even when the change in distribution from layout parasitics is significant.

Table 3.12: Prior work and AutoCkt layout performances for 8-bit and 10-bit pipeline ADC specifications with load capacitance varying from $8 - 30fF$.

| Metric | 8-bit | AutoCkt | [16] | [63] | 10-bit | AutoCkt | [16] | [63] |
|---|---|---|---|---|---|---|---|---|
| UGBW ($GHz$) | $> 0.98$ | 3.0 | 4.46 | 5.8 | $> 1.2$ | 4.18 | 4.9 | 5.8 |
| Gain ($dB$) | $> 54$ | 59.0 | 44.0 | 57.6 | $> 60$ | 61.6 | 53.5 | 61.9 |
| Noise ($\mu V_{rms}$) | $< 750$ | 585.7 | 421.1 | 414.7 | $< 380$ | 519 | 390.0 | 423.3 |
| Power ($\mu A$) | min | 492 | 501 | 450 | min | 589 | 556 | 581 |
| PM ($°$) | $> 60$ | 71.1 | 23.9 | 13.1 | $> 60$ | 70.4 | 28.7 | 22.2 |
| Tset ($ns$) | 1.7 | 1.44 | 7.23 | 15.7 | 1.7 | 1.37 | 7.1 | 10.4 |
| Swing ($V$) | 0.80 | 0.88 | 0.98 | 0.62 | 0.8 | 0.89 | 0.80 | 0.83 |

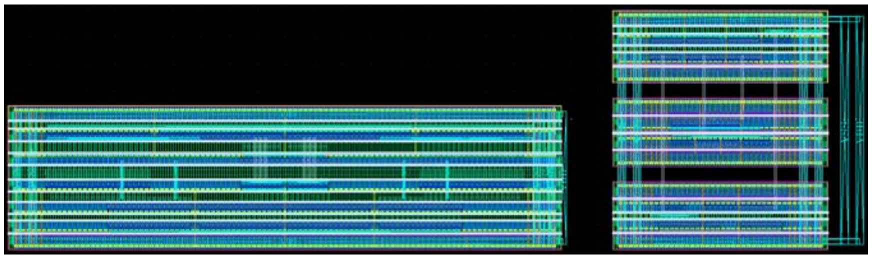


Figure 3.18: BAG-generated LVS-passed layout for the core and biasing folded-cascode circuit for an obtained performance.

Table 3.13: Required Simulation Count (RSC) for AutoCkt with varying seeds (average min/mean/max) and two prior works

| Experiment | Schematic RSC | RSC PEX | Obt. Layout Target? |
|---|---|---|---|
| AutoCkt, Seed 1 | 4 / 30 / 40 | 4 | Yes |
| AutoCkt, Seed 2 | 3 / 27 / 40 | 4 | Yes |
| AutoCkt, Seed 3 | 3 / 25 / 40 | 4 | Yes |
| [19] | - | 23 | Yes |
| [16] (schematic) | 10012 | - | No |
| [63] (schematic) | 58844 | - | No |

### Pipeline ADC Target Specifications

We select the two target specifications for the 8-bit and 10-bit pipeline ADC (Table 3.5) and input them to the trained agent during deployment. The results show that the framework obtains LVS-passed designs that meet all target metrics for the 8-bit ADC, and all metrics for the 10-bit ADC when noise variability is taken into account. The obtained performances are summarized in Table 3.12, and the BAG-generated folded-cascode amplifier and biasing layout is shown in Figure 3.18. Table 3.12 also shows the layout performances obtained by two prior works run with schematic simulations, whose converged designs are then taken and simulated with layout in BAG. Despite both works obtaining designs that meet the target in schematic, they fail to meet or obtain valid performances once parasitics are considered, with phase margin, gain and/or settling time degrading significantly. We note that the most efficient schematic RSC solution from [33] would take almost 13.9 days to converge using layout parasitic simulations with the same RSC.

We note that the RL algorithm handles the distribution difference between schematic and layout by requiring additional simulations in schematic to meet the target, on average taking 30 schematic simulation steps and 4 parasitic steps across the 60 tested target specifications, shown in Table 3.13. This approach reliably reaches target specifications in layout, with state-of-the-art PEX simulation RSC, beating [19] by almost 6× in a more complex circuit topology. We note that prior work would have required too many RSCs to deploy with layout simulations, as their RSCs in schematic are too high.

### PEX Corner Simulation Results

To determine if AutoCkt reaches design points that are robust enough once process corners are considered, corner analyses is run for each of the 60 performances the agent obtained. To understand the validity of these results, which are shown in Table 3.14, we calculate two metrics: 1) once corner simulation is completed, how many of the 60 design points still have valid results, meaning the metrics are within the same region as the nominal corner, and 2) do the designs that satisfy the 8-bit and 10-bit pipeline ADC requirement still meet these targets once corners are considered.

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

We also show the corner simulation results for the previous 8-bit and 10-bit ADC designs in Tables 3.15 and 3.16. For both designs, the target specifications are satisfied for every metric, with the exception of noise in the 10-bit ADC design. We again posit that the noise metric is an estimated value. We run Monte Carlo simulations on these designs as well, but find that the results varied considerably.

To incorporate process corner/Monte Carlo PVT simulations, we propose using accurate estimation tools such as [45, 65, 66] to predict these variations during training with schematic simulations, giving the agent the mean and standard deviation of each metric performance as part of the state, or use a similar deployment generalization methodology to run process corner/Monte Carlo simulations only when the agent has obtained a valid schematic and layout performance design. If the results of these simulations do not meet a predefined acceptable variation, the worst performance is given back to the agent to once again go through the schematic and layout sizing procedure.

Table 3.14: FF, SS, SF, SS corner simulation results showing minimum, mean, and maximum obtained metric performances across 60 agent-obtained designs.

| Metric | **FF** | | | **FS** | | | **SF** | | | **SS** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max |
| $f_t(GHz)$ | 2.7 | 3.6 | 5.1 | X | 3.3 | 4.8 | 2.75 | 3.6 | 5.0 | X | 3.3 | 4.6 |
| $A_{vo}(dB)$ | 54 | 57 | 59 | X | 54 | 60 | 56 | 60 | 62 | X | 54 | 62 |
| $V_{ns}(\mu V_{rms})$ | 509 | 538 | 702 | 452 | 525 | 691 | 495 | 580 | 681 | 467 | 514 | 619 |
| $P(\mu A)$ | 457 | 598 | 611 | 452 | 16$m$ | 957$m$ | 464 | 543 | 617 | 442 | 14$m$ | 849$m$ |
| $pm(°)$ | 63 | 70 | 83 | 60 | 69 | X | 62 | 70 | 82 | 58 | 68 | 80 |
| $t_{set}(ns)$ | 0.87 | 1.5 | 3.4 | 0.37 | 1.13 | 3.16 | 0.85 | 1.46 | 3.3 | 1.18 | 6.74 | X |
| $V_{pp}(V)$ | 0.66 | 0.84 | 0.95 | 0.64 | 0.77 | 0.86 | 0.78 | 0.93 | X | X | 0.82 | 0.98 |

Table 3.15: 8-bit corner simulation results.

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

Table 3.16: 10-bit corner simulation results.

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

### Comparison to Prior Analog Sizing Work

We compare AutoCkt to prior analog sizing works in Table 3.17. In this comparison, we take the best-performing, most-complex circuit topologies tested by each prior work, and include their circuit complexity measured by the number of parameters in the circuit and number of metrics tested, whether parasitics are considered, whether a parasitics model is used, process corners are considered, and the RSC. We show that this work tests on a complex circuit while considering true layout parasitic simulations, and has one of the lowest deployment RSC values.

We also compare one of the performances reached by AutoCkt to prior work that have designed folded-cascodes, shown in Table 3.18. Not only does the agent reach comparable gain and at higher bandwidth, it does so with less power.

## 3.4 Open-source code and framework

The AutoCkt framework, code, and schematic results presented in [19] can be found on GitHub at https://github.com/ksettaluri6/AutoCkt. Due to licensing constraints, the code

Table 3.17: Comparison table for prior analog automation works.

| | [44] | [46] | [45] | [25] | [39] | [16] | **This Work** |
|---|---|---|---|---|---|---|---|
| Algorithm | Metamodel | Metamodel | Metamodel | GCN+RL | RL+start | NN+GA | RL |
| Technology | 45*nm* | 90*nm* | 90*nm* | 180*nm* | 6*nm* | 14*nm* | 16*nm* FF |
| Complexity (# params, #metrics) | $6, 2$ | $5, 1$ | $6, 2$ | $25, 5$ | $4, 2$ | $17, 9$ | $29, 7$ |
| Training cost | Low | Low | Low | High | Medium | Medium | High |
| Considers parasitics | Yes | Yes | Yes | No | No | Yes | Yes |
| Uses parasitics model | Yes | Yes | Yes | No | No | No | No |
| Process corners | No | Yes | Yes | No | Yes | No | Yes* |
| RSC | 2 | 2 | 2 | 10000 | 2609 | 435 | 4 |

* Validated on process corners, but not trained

Table 3.18: Comparison table for folded-cascode topologies.

| Parameter | [1] | [2] | [3] | [4] | [5] | **Work** |
|---|---|---|---|---|---|---|
| Process | $65nm$ | $0.18\mu m$ | $0.18\mu m$ | $0.5\mu m$ | $0.18\mu m$ | $16FF$ |
| VDD | $0.5V$ | $5V$ | $1.8V$ | $+-1V$ | NA | $1.0V$ |
| Cap. load | NA | $10pF$ | $2pF$ | $70pF$ | $5.6pF$ | $14fF$ |
| $t_{set}$ (rise/fall) | $892/820ns$ | NA | NA | $96ns/74ns$ | $20.7ns$ | $1.23ns$ |
| $A_{vo}$ | $64.2dB$ | $89dB$ | $48dB$ | $81.7dB$ | $52.6dB$ | $60.3dB$ |
| $pm$ | $61^o$ | $60^o$ | NA | $60^o$ | $83.6^o$ | $68.2^o$ |
| $f_t$ | $2.1MHz$ | $17MHz$ | $40MHz$ | $4.75MHz$ | $70.7MHz$ | $4.15GHz$ |
| $V_{ns}(nV/\sqrt{Hz})$ | NA | $9.5@10kHz$ | $99.35\mu$ | $35@1MHz$ | $53.2\mu$ | $549\mu$ |
| $V_{pp}$ | 705mV | rail-to-rail | 1.42V | NA | NA | $0.86V$ |
| Topology* | FC+CS | FC+AB | FC+AB | FC | FC | FC+AB |
| Power | $150\mu W$ | $790\mu A$ | $800\mu A$ | $80\mu W$ | $800\mu A$ | $498\mu A$ |

*AB = Class AB

only interfaces with an open-source circuit simulator, NGSpice.

### 3.4.1 Setup

The setup requires Anaconda. In order to obtain the required packages, run the command below from the top level directory of the repo to install the Anaconda environment, and activate the environment:

- conda env create -f environment.yml
- source activate autockt

You might need to install some packages further using pip if necessary. To ensure the right versions, look at the environment.yml file. NGspice 2.7 needs to be installed separately, via https://sourceforge.net/projects/ngspice/files/ng-spice-rework/ol/27/. Page 607 of the PDF manual on the website has instructions on how to install. Note that you might need to remove some of the flags to get it to install correctly for your machine.

### 3.4.2 Code flow

The code setup is shown in Figure 3.19.

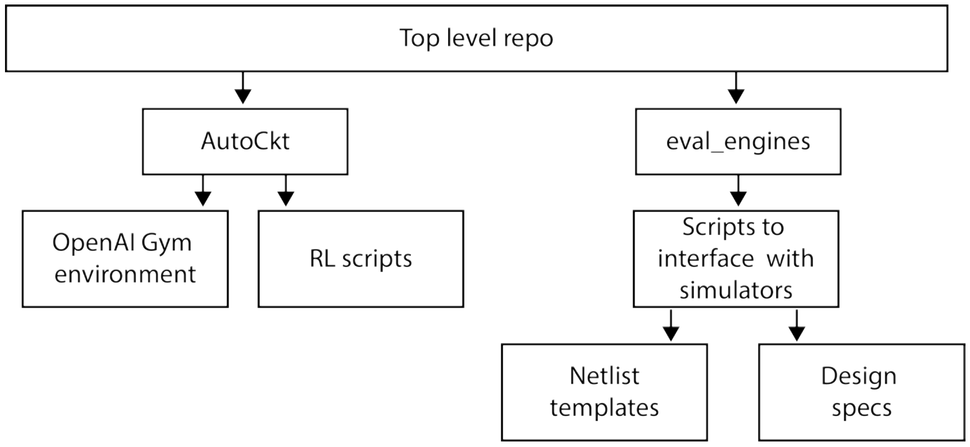


Figure 3.19: Flow-chart for AutoCkt framework on Github.

The top level directory contains two sub-directories:

- AutoCkt: contains all of the reinforcement code
  - val_autobag_ray.py: top level RL script, used to set hyperparameters and run training
  - envs directory: contains all OpenAI Gym environments. These function as the agent in the RL loop and contain information about parameter space, valid action steps and reward.
- eval_engines: Contains all of the code pertaining to simulators
  - ngspice: this directory runs all NGSpice related scripts.
  - specs_test: a directory containing a unique yaml file for each circuit with information about design specifications, parameter ranges, and how to normalize.
  - script_test: directory with files that test functionality of interface scripts

### 3.4.3 Training

Make sure that you are in the Anaconda environment. Before running training, the circuit netlist must be modified in order to point to the right library files in your directory. To do this, run the following command:

- python eval_engines/ngspice/ngspice_inputs/correct_inputs.py

To generate the design specifications that the agent trains on, run:

- python autockt/gen_specs.py –num_specs ##

The result is a pickle file dumped to the gen_specs folder. To train the agent, open ipython from the top level directory and then:

- run autockt/val_autobag_ray.py

The training checkpoints will be saved in your home directory under ray_results. Tensorboard can be used to load reward and loss plots using the command:

- tensorboard –logdir path/to/checkpoint

To replicate the results from the paper, num_specs 350 was used (only 50 were selected for each CPU worker). Ray parallelizes according to number of CPUs available, that affects training time.

## 3.5 AutoCkt analyses and conclusion

After training and deploying AutoCkt on a two-stage amplifier, two-stage amplifier with negative $g_m$ load, and the two-stage folded-cascode, we have found several crucial factors in considering this machine-learning sizing approach. In particular:

- The choice of target specifications is important in determining how well the agent generalizes during deployment. It appears that the agent performs better in regions where it has already explored and trained, versus areas that are vastly outside the range of the given target design specifications. This is not surprising, as the agent is given rewards for its performance on the data it explicitly trains off of.
- The random selection of target specifications within a range is not reflective of any actual circuit topology constraints, making accuracy metrics after, as seen in this paper, inaccurate to solely consider. To obtain better accuracy values in the future, there should be a better method of generating design specifications that more intuitively follow the specific circuit topology the RL algorithm is trained on.
- The choice of reward function is robust across topology differences, even when the metrics themselves are vastly different.
- Despite not being trained on corner simulation results, the agent still obtains feasible design points when these simulations are run. The robustness of this methodology can be improved by incorporating corner simulations during the training process.

In this chapter we expand the results on the reinforcement-learning tool that sizes analog circuits, AutoCkt [19], to show that this framework is scalable, accurate, and practicable. We test the framework on a two-stage transimpedance amplifier and two-stage folded-cascode with biasing, and demonstrate that this tool achieves state-of-the-art run-time RSC, at least 38× more efficient than prior work. We show that the algorithm is robust, generalizes to an environment that considers layout parasitics, and can handle the large amount of error present between post-layout and schematic-only simulations. With the combined schematic and layout-parasitic simulation approach, achieved by using the Berkeley Analog Generator, We find that this approach requires, on average, 4 layout simulations to converge to a performance that meets a target specification. In addition, the obtained LVS-passed folded-cascode performances meet target 8-bit and 10-bit pipeline ADC designs that are valid when process corners are considered. We also conduct extensive analysis on where this framework fails, and why this machine-learning approach can be practically incorporated to an existing analog-sizing flow.

# Chapter 4

# Reinforcement Learning for Nonlinear Circuit Topologies

In this chapter, we explore two circuit topologies in a pipeline ADC application that require more design effort compared to other conventionally well-defined circuits: a ring amplifier (RAMP) and a comparator. We demonstrate that the reinforcement-learning framework achieves target specifications in schematic simulations for the comparator and both schematic and layout simulations for the RAMP, and show that the automatically generated designs achieve comparable, and even better performances when compared against a manually sized design from an expert. In addition, we show the agent's pareto curves and more analysis on the efficacy of the algorithm to function on these more complex circuits. Specifically,

- We show that the established sizing procedures for both circuits are heuristic and manual, increasing the number of iterations and end-to-end design time,
- the trained RL agent converges to many different target specifications with, on average, just five schematic simulations for the comparator,
- the trained agent converges to many target specifications with, on average, two layout simulations for the ring amplifier topology, and
- compared to an expert, the agent achieves 12% and 18% better designs for the comparator and ring amplifier, respectively.

## 4.1 Introduction

Comparators are crucial to ADC design, as they dictate the speed and precision of the ADC, and are responsible for sensing the difference between differential inputs and outputting a logical signal corresponding to the polarity of the input difference [67]. Though many different topologies exist, strong-arm comparators in particular provide several unique benefits: 1) they do not draw static power, 2) have rail-to-rail output swing, 3) require one clock-phase, and 4) sources of input offset are primarily due to the input pair transistors. The design of this type of comparator is defined in several manual procedures [68, 69], and is selected for testing on this automated sizing methodology primarily to assess the efficacy of the tool in matching the methodology developed by an expert analog designer.

Ring amplifiers [70] are scalable, modular amplifiers that use a cascade of dynamically stabilized inverters to perform accurate amplification, and are specifically applicable in modern high-linearity, high-bandwidth pipeline ADC designs because they can provide rail-to-rail swing, charge large capacitive loads, and scale with process technologies. Whereas prior topologies used in the residue amplification step often face power-efficiency issues, RAMPs, when paired with a first-order gain calibration step, can provide a power-effective solution for high linearity and speed in nanoscale CMOS [71]. Designing RAMPs, however, is not as straightforward as other well-defined amplifier topologies, particularly because multiple testbench simulations must be considered in parallel to successfully size the topology to meet a target specification. In particular, the sizing methodology involves considering slewing, settling and stabilizing modes of operation, covered in more detail in Section 4.3.

This chapter is organized as follows:

- Section 4.2 discusses the strong-arm comparator, sizing approach, and schematic results on the circuit topology.
- Section 4.3 introduces the RAMP topology, presents the traditional methodology to size the amplifier, and demonstrates AutoCkt's results on the circuit in both schematic and post-extracted layout simulations.
- Section 4.4 concludes with additional analyses and direction for future work.

## 4.2 Strong-arm comparator

Figure 4.1 shows the selected strong-arm comparator topology [67]. The circuit itself contains few circuit parameters to modify, with with only 11 total transistors.

### 4.2.1 Comparator basics

The comparator functions by first precharging the four internal nodes in the circuit to the power supply, $VDD$. The $CK$ signal then goes high, causing the internal nodes that connect to the drains of $M_1$ and $M_2$ to drop in voltage proportional to the difference between the inputs $V_{in1}$ and $V_{in2}$. Once the $M_1$ and $M_2$ drain voltages drop to $VDD - V_{th3,4}$, $M_3$ and $M_4$ are activated, causing their drain voltages to drop also until $M_5$ and $M_6$ are activated. The output voltage terminals then either receive $VDD$ and ground, or vice versa, completing the comparator operation [68].

### 4.2.2 Manual comparator design

The design of this comparator begins by heuristically determining sizes of the signal path transistors ($M_1$, $M_2$, $M_3$, $M_4$, $M_5$, $M_6$) to meet a given voltage offset. Nominally the threshold voltage mismatch, $\Delta V_{th1,2}$, is:

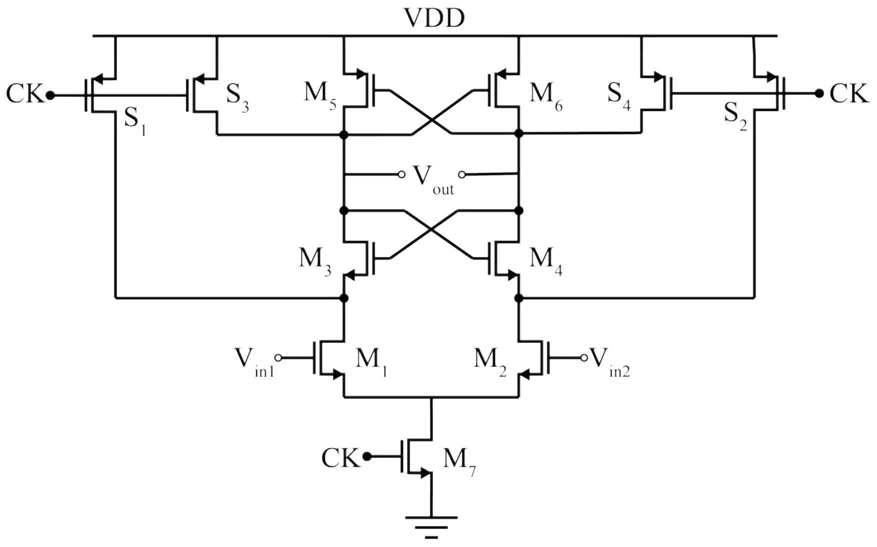


Figure 4.1: Strong-arm comparator schematic.

$$\Delta V_{th1,2} = \frac{A_{Vth}}{\sqrt{WL_{1,2}}} \tag{4.1}$$

where $A_{Vth}$ is a constant dictated by the technology node. $\Delta V_{th1,2}$ is then determined by choosing an initial width and length for $M_1$ and $M_2$.

Next, the $M_7$ tail transistor is sized to draw enough current, and operates in the deep triode regime. Transistors $M_3$ and $M_5$ are sized heuristically, as the offset contributed by them is not significant, thus the authors in [67] state that this threshold mismatch can be nominally divided by a factor between $3 - 5\times$. A similar philosophy is used to select the widths for $M_5$ and $M_6$; these transistors affect the comparator's speed, however, which then requires further iteration to assess this effect once the design has been simulated. Finally, the reset switches $S_1$, $S_2$, $S_3$, and $S_4$ are sized based on their requirement to pull their drain voltages to $VDD$ under a certain, pre-constrained time.

Once the design is initially sized, the strong-arm comparator is simulated, upon which further analysis is conducted to meet the offset and speed requirements. We note that several prior works have discussed this sizing procedure [69], [72], but all purportedly rely on heuristics, iteration, and intuition in order to successfully design this circuit.

### 4.2.3 Comparator RL setup

We move to discuss the tunable parameters and target specifications that interface the strong-arm comparator to the reinforcement-learning algorithm.

**Parameters**

The circuit is tested in Intel $22nm$ technology, with parameters that correspond to the width of the MOS devices in Figure 4.1. Matching is incorporated as a constraint, meaning that the total tunable parameters given to the agent are the four pairs of devices ($M_1$ and $M_2$, $M_3$ and $M_4$, $M_5$ and $M_6$), the tail transistor, $M_7$, and a single switch parameter that sizes $S_1$ to $S_4$ equally. Each action space parameter range varies starting from $2ww_b$ to $40ww_b$ in increments of 2, where $w$ is the base number of fingers, and $w_b$ is the base width increment, which for this technology, is $45nm$.

**Target specifications**

The target metrics are noise ($V_{noise}$), delay ($t_d$), input capacitance ($C_{in}$), and dissipated power ($P$). 100 target specifications are given to the agent, sampled around an expert-designed schematic performance, and summarized in Table 4.1.

Table 4.1: Performance range sampled during training for strong-arm comparator.

| Metric | Minimum | Maximum |
|---|---|---|
| $t_d$ $(ps)$ | 1 | 100 |
| $V_{noise}$ $(\mu V_{rms})$ | 100 | 600 |
| $C_{in}$ $(fF)$ | 1 | 10 |
| $P$ $(\mu A)$ | 20 | 200 |

### 4.2.4 Results on training and deploying with schematic simulations

The results of training the RL algorithm with the strong-arm comparator are discussed in this section. Figure 4.2 shows that the mean trajectory reward increases to the stopping condition of zero, meaning that the agent successfully reaches many target specifications.

We then deploy the agent on schematic simulations only, and the summary of minimum, mean, and maximum obtained performances for each target metric is shown in Table 4.2 for 54 designs. The range in metrics not only suggests that the agent understands different regions of the design space, but is also able to understand the intricacies in strong-arm comparator design, requiring just over 10000 simulation steps to train, but only, on average, 5 steps to deploy.

To understand how these schematic performances compare to those of an expert, Table 4.3 shows an expert-designed and agent-obtained performance for the same target specification. This table shows that the agent can not only achieve a comparable performance, but also reaches better regions of operation compared to the expert. This, of course, is only an initial study with schematic-only simulations. There is additional research that must be completed

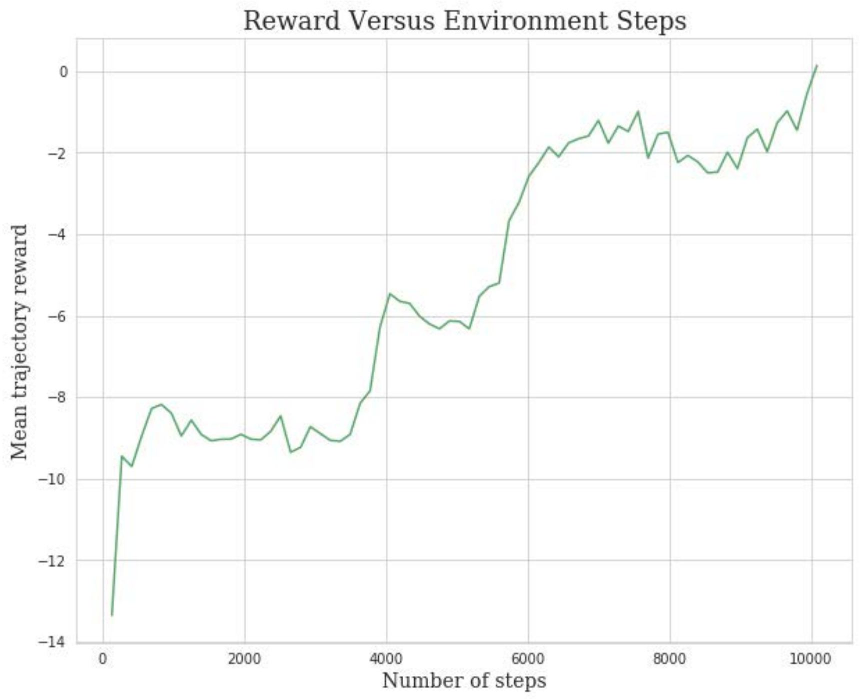


Figure 4.2: Mean episode reward for strong-arm comparator.

in order to fully understand the feasibility and practicality of this RL tool for the strong-arm comparator, which includes running with layout parasitics and process corner simulations.

For this circuit, there was also some difficulty in training the tool to take more than one iteration of training to converge - as the parameters and target specification space were limited. To fix this, the range of parameters was increased, in addition to reducing several RL hyperparameters, including training batch size and horizon length. We posit that this issue can be fixed by increasing target metrics, perhaps by incorporating voltage offset.

The quality of results achieved when the RL agent trained and converged to a positive reward in one training iteration was low, as expected, due to the initial random sampling of designs and trajectories during training. We emphasize that this RL methodology is only successful in understanding the relationship between parameters and target specification when there is a clear path to improving reward during multiple iterations of training.

## 4.3 Ring amplifier

In this section, we discuss RAMP basics, the equations and analysis used in manual RAMP design, the interface between the selected RAMP topology and the RL framework, and the results associated with both schematic and layout parasitic simulations.

Table 4.2: Performance range obtained by agent during training.

| Metric | Minimum | Mean | Maximum |
|---|---|---|---|
| $t_d$ $(ps)$ | 46 | 65 | 95 |
| $V_{noise}$ $(\mu V_{rms})$ | 20 | 35 | 76 |
| $C_{in}$ $(fF)$ | 2.6 | 6.5 | 112 |
| $P$ $(\mu A)$ | 134 | 195 | 291 |

Table 4.3: Comparison for expert-designed and agent-obtained performances for the same target specification.

| Metric | Expert | Agent | Percent Difference |
|---|---|---|---|
| $t_d$ $(ps)$ | 88 | 82 | $-6.8$ |
| $V_{noise}$ $(\mu V_{rms})$ | 270 | 233 | $-13.7$ |
| $C_{in}$ $(fF)$ | 2.57 | 3.33 | 29.6 |
| $P$ $(\mu A)$ | 20 | 20 | 0 |

### 4.3.1 RAMP basics

The RAMP topology we have selected, shown in Figure 4.3, consists of three dynamically stabilized inverters that provide multi-stage amplification, with an anti-parallel CMOS configuration in the second stage [70]. RAMP operation can be divided into three distinct parts: 1) the amplifier initially charges the capacitive load, $C_L$, in an unstable, high slew-rate mode, 2) the RAMP then dynamically adjusts the biasing so that a dominant pole appears at the output, increasing the phase margin, and 3) finally locks to a stable region with positive phase margin [71]. The key feature in obtaining stability is the induced dead-zone (DZ) voltage controlled by $V_{B,H}$ and $V_{B;L}$ that drives the output devices $M_{P3}$ and $M_{N3}$ with different versions of the input signal, determining both the static and dynamic properties of the amplifier.

#### Static analysis

We first analyze the amplifier's gain, and discuss how $V_{DZ}$ impacts this metric. Specifically, total open loop gain is obtained by calculating the gain at the first stage and multiplying it with the combined gain in the second and third stages, as outlined by [71]. The stage one gain, $A_{s1}$ is:

$$A_{s1} = (g_{m,N1} + g_{m,N2}) * (r_{o,N1} || r_{o_N 2})$$

where $g_{m,X}$ and $r_{o,X}$ are the $g_m$ and $r_o$ for transistor $X$. The gain, $A_{s2,3}$, for stages two and three is:

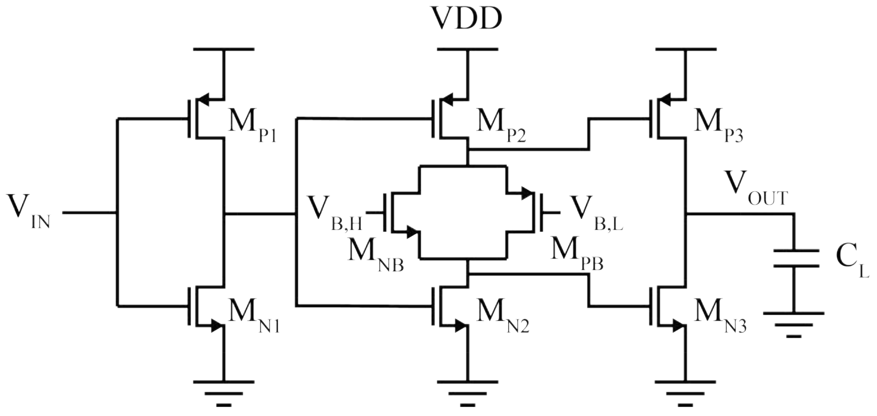


Figure 4.3: Ring amplifier topology proposed in [71]

$$\begin{aligned}A_n = (g_{m,P2} + g_{m,N2})(g_{m,P3}(g_{m,NB} * r_{o,PB}||r_{o,NB} + 1) + g_{m,N3}(g_{m,PB} * r_{o,PB}||r_{o,NB} + 1)) \\ + g_{m,P3}g_{m,P2}\frac{r_{o,PB}||r_{o,NB}}{r_{o,N2}} + g_{m,N3}g_{m,N2}\frac{r_{o,B}}{r_{o,P2}}\end{aligned} \tag{4.2}$$

$$A_{d1} = r_{o,P2}(g_{m,PB}r_{o,B} + 1) + r_{o,PB}||r_{o,NB} + r_{o,P2}(g_{m,PB}r_{o,B}) \tag{4.3}$$

$$A_{d2} = r_{o,P2} * r_{o,N2} * r_{o,P3}||r_{o,N3} \tag{4.4}$$

$$A_{s2,3} = \frac{A_n}{A_{d1}/A_{d2}} \tag{4.5}$$

Making the total gain, $A_{tot}$, equal to:

$$A_{tot} = A_{s1} * A_{s2,3} \tag{4.6}$$

This gain expression is relatively unintuitive due to the complexity, despite the topology being simple, leading to more difficulty in the design process. In addition, the relationship between the deadzone voltage, $V_{DZ}$, which is an important part of the sizing the RAMP, to this gain expression is obtained by conducting sweeps that lead to several heuristics. Specifically, [71] finds that $V_{DZ}$ and $A_{tot}$ are inversely related, with $V_{DZ}$ having no significant impact on static linearity.

## Dynamic analysis

Static analysis captures only the steady-state behavior of the circuit, and both DC and dynamic testbenches are needed to ensure reaching and locking to the steady-state region.

We calculate the gain-bandwidth $f_{GBW}$, once the output node is the dominant pole $dp$, with frequency $f_{dp}$:

$$f_{dp} = \frac{1}{2\pi * r_{o,P3}||r_{o_N3} * C_L} \tag{4.7}$$

The gain-bandwidth, $f_{GBW}$ is then:

$$f_{GBW} \approx \frac{1}{2\pi} A_{S1} A_{S2} \frac{g_{m,P3} + g_{m,N3}}{C_L} \tag{4.8}$$

where $A_{S1}A_{S2}$ can be split into their partial gains $A_{S1,S2,P}$ and $A_{S1,S2,N}$:

$$A_{P(N)} = -r_{o,P2(N2)}[g_{m,P2(N2)} r_{o,PB}||r_{o,NB} + (g_{m,P2} + g_{m,N2}) r_{o,N2(P2)}] \tag{4.9}$$

$$A_{P(N),den} = r_{o,P2}(g_{m,BP} r_{o,PB}||r_{o,NB} + 1) + r_{o,PB}||r_{o,NB} + r_{o,N2}(g_{m,NB} r_{o,PB}||r_{o,NB} + 1) \tag{4.10}$$

$$A_{S1,S2,P(N)} = A_{S1} * \frac{A_{P(N)}}{A_{P(N),den}} \tag{4.11}$$

This expression is complex, with [71] and [73] mentioning that the deadzone voltage impacts it significantly as well, by lowering $f_{GBW}$ but improving phase margin upon increases to $V_{DZ}$.

### Manual RAMP design

Several works outline how to size a RAMP, though the procedure is relatively heuristic in nature. For example, the proposal outlined by [70] discusses placing the ring amplifier in the target application, with realistic timing and feedback conditions, and doing the following: 1) initialize ring amplifier with extra bandwidth, 2) size the output stage for worst-case slewing/settling, 3) reduce the sizes of the front stages for power efficiency, and 4) iterate steps 2-3 if necessary. [70] emphasizes, however, that the design procedure is heavily dependent on the application and optimization priorities.

Other methods like [74] however, suggest 1) sizing the unit inverter first by choosing nominal values for the NMOS (which should be small for the first two stages), and selecting PMOS sizes that are two times the width of the NMOS, assuming some resistance and offset voltage, 2) running a unit-inverter optimizer for all inverter stages, 3) sizing the top-level, 4) assessing validity at the top-level, and then based on this feedback iterate from 1) accordingly.

These prior methods appear to be heuristic in nature, and require iterative loops in order to successfully size the topology. In this formulation, we use the reinforcement-learning framework to size the RAMP by considering AC and transient testbenches.

## 4.3.2 RAMP Autockt setup

We discuss the parameters and specifications that interface the RAMP to Autockt.

### Parameters

The tunable parameters for this topology include width of the MOS devices, the two bias voltages $V_{B,H}$ and $V_{B_L}$, and load capacitance. Each parameter range is derived from the BAG layout generator used to consider layout parasitics, and has 11 unique values to select from. We note that load capacitance is a tunable parameter to allow the agent to size the RAMP for different load conditions, and is fixed during deployment. We focus on uniform channel-length devices when parasitics are included, limited by the simplicity of the BAG layout generator.

### Target specifications

We select the following target metrics, suggested by [70]:

– open-loop gain, $A_{vo}$
– phase margin, $pm$
– unity-gain frequency, $f_t$
– settling time, $t_{set}$
– dissipated power, $P$

The settling time measurement is obtained by connecting the ring amplifier in unity feedback, indicating the stability of the amplifier, as well as constrain the design to the RAMP only. The target specifications generated during training are selected from the range in Table 4.4, based on roughly sampling regions around an expert-designed performance.

Table 4.4: Performance range sampled during training for ring amplifier.

| Metric | Minimum | Maximum |
|---|---|---|
| $A_{vo}$ $(dB)$ | 25 | 70 |
| $f_t$ $(GHz)$ | 0.1 | 1.4 |
| $pm$ $(°)$ | 60 | 75 |
| $t_{set}$ $(ns)$ | 0.1 | 100 |
| $P$ $(\mu A)$ | 500 | 1100 |

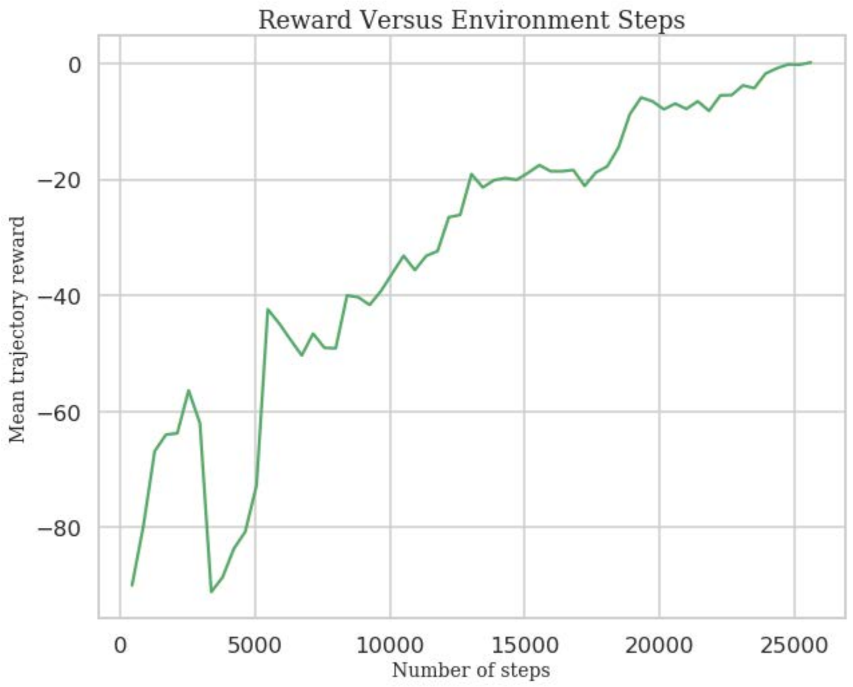


Figure 4.4: Mean reward per trajectory for the ring amplifier.

Table 4.5: Required simulation count (RSC) to converge and number of specifications reached versus tested: ring amplifier.

| Metric | RSC | # Specs Reached / # Specs Tested |
|---|---|---|
| AutoCkt (training) | 25620 | 80/100 |
| AutoCkt (deploy) | 12 | 260/325 |

### 4.3.3 Results on training and deploying with schematic simulations

The mean reward per trajectory for each iteration of training (represented as number of simulations or steps) is used to assess whether the algorithm trained successfully with schematic simulations. Figure 4.4 shows that this reward reaches zero after training has completed, meaning that the agent, on average, has learned to reach many target specifications.

To understand the distribution of schematic simulations the agent is able to reach, the trained agent is deployed with schematic simulations only. Table 4.5 shows the number of targets the agent reaches in training and deployment, and demonstrates that the algorithm performs equally well in both cases. In addition, during deployment, the agent requires, on average, just 12 simulations to converge. Table 4.6 summarizes the range in agent-obtained performances, and shows not only that the agent reaches valid AC and transient results, but does so with a variety of performances.

Table 4.6: Obtained schematic simulation performance ranges by agent.

| Metric | Minimum | Mean | Maximum |
|---|---|---|---|
| $A_{vo}$ $(dB)$ | 3.01 | 22.2 | 56.7 |
| $f_t$ $(GHz)$ | 4.2 | 11.4 | 15.7 |
| $pm$ $(°)$ | 60.3 | 82.1 | 140.7 |
| $t_{set}$ $(ns)$ | 0.07 | 0.09 | 0.45 |
| $P$ $(\mu A)$ | 865 | 1200 | 1350 |

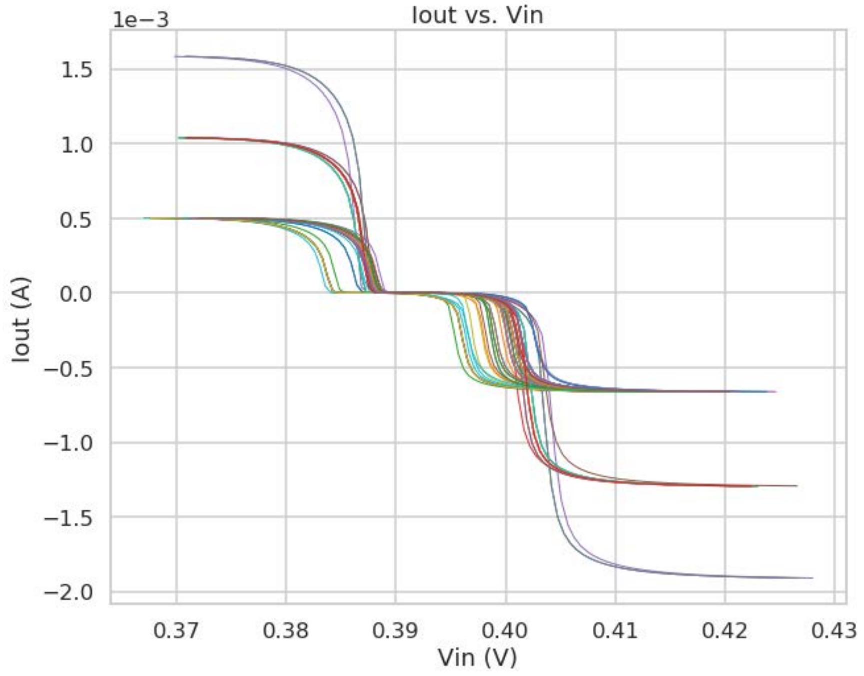


Figure 4.5: Output current versus input voltage schematic simulation plot for the ring amplifier for different designs.

In addition to the selected target metrics, we assess several RAMP-specific characteristics. Figure 4.5 plots the output current versus input voltage for the amplifier, and shows that the agent obtains differing design points while maintaining a flat deadzone area. This is important, as the deadzone width is positively correlated to stability. Figure 4.6 plots the transient results of the ring amplifier in unity-gain feedback, and shows that the obtained designs are stable. Lastly, Figure 4.7 shows the output voltage over input voltage, the $V_{B,H}$

and $V_{BL}$ voltages versus input voltage, and the gain over output voltage, which verify the deadzone voltage width and linearity.

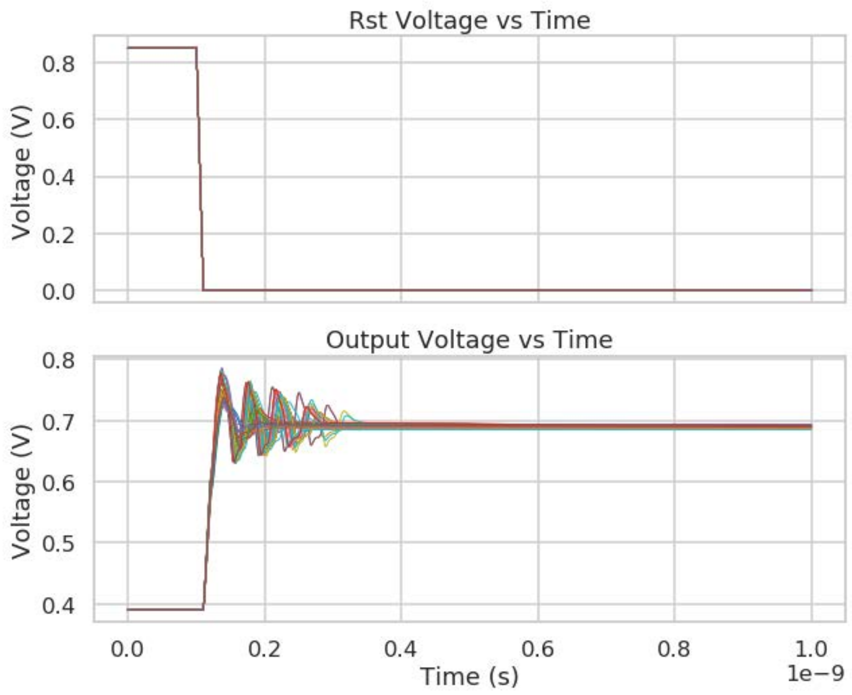


Figure 4.6: Schematic simulation results for reset and output voltage versus time for the ring amplifier for different designs.

## 4.3.4 Results on deploying with layout simulations

In this section, we discuss the results of the reinforcement-learning agent when layout parasitics are considered. To do this, we use a RAMP generator created in BAG that automatically creates the schematic, layout and testbenches for this topology. The end-to-end simulation time using BAG is nearly 4× more than simulating using just schematic simulations. As we will later show, this prevents prior RSC-inefficient work from running in an environment that solely does BAG simulations.

Before deploying the agent, we would like to assess the differences between schematic and post-extracted layout (PEX) simulations. To do so, we select design points and simulate in both schematic and PEX for the same set of circuit parameters, and plot the differences in each target metric, shown in Figure 4.8. We see that the degradation in performance is significant for all target metrics, with gain having a distinct nonlinear relationship, making it difficult for prior layout parasitic estimation models to correctly predict this effect.

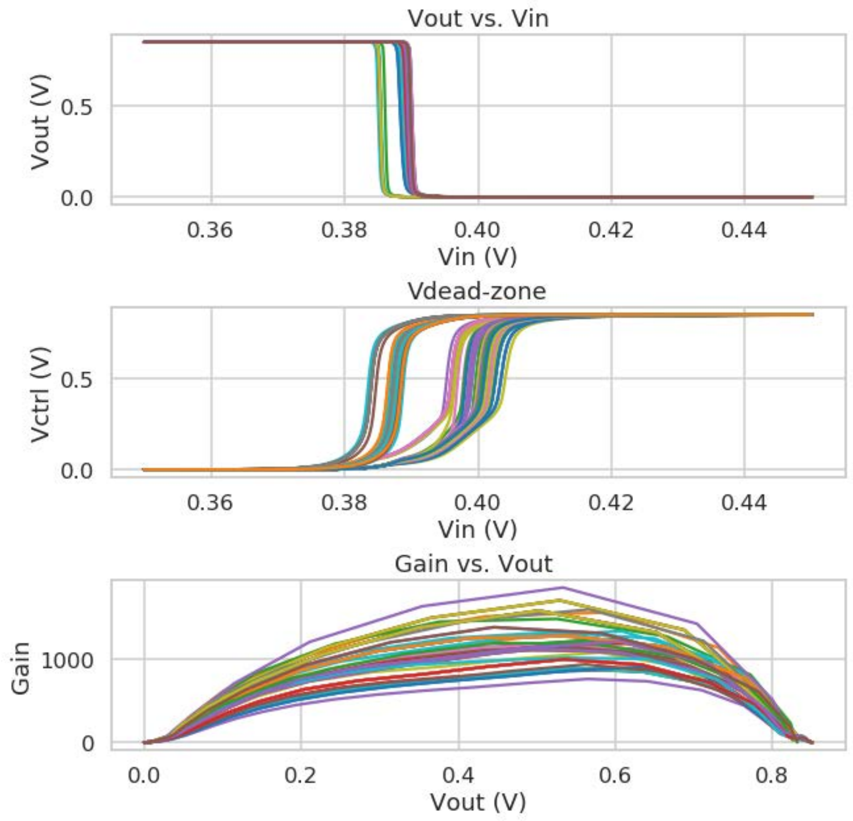


Figure 4.7: Obtained schematic performance plots of output versus input voltage (top), deadzone versus input voltage (middle), and gain versus output voltage (bottom) for the ring amplifier for different designs.

### Impact of layout parasitics to performance

We deploy the schematic trained agent with in an environment that considers layout parasitics. The obtained performances are summarized in Table 4.7, for 40 different designs, and show that the agent obtains a wide range of performances. The corresponding RAMP-specific characteristics are plotted in Figures 4.9, 4.10 and 4.11, and show that the designs are stable, with wide deadzone voltages and high linearity. We note degradation in performance between schematic to layout simulations, as expected, but find that the agent's designs are still valid and reach target specifications.

We compare the agent's performance against an expert when given the same target specification. The results are summarized in Table 4.8, and show that the algorithm can obtain a performance that is better than an expert in every metric. This is important, as it allows the designer to potentially discover new and previously unexplored regions in the design space using this algorithm.

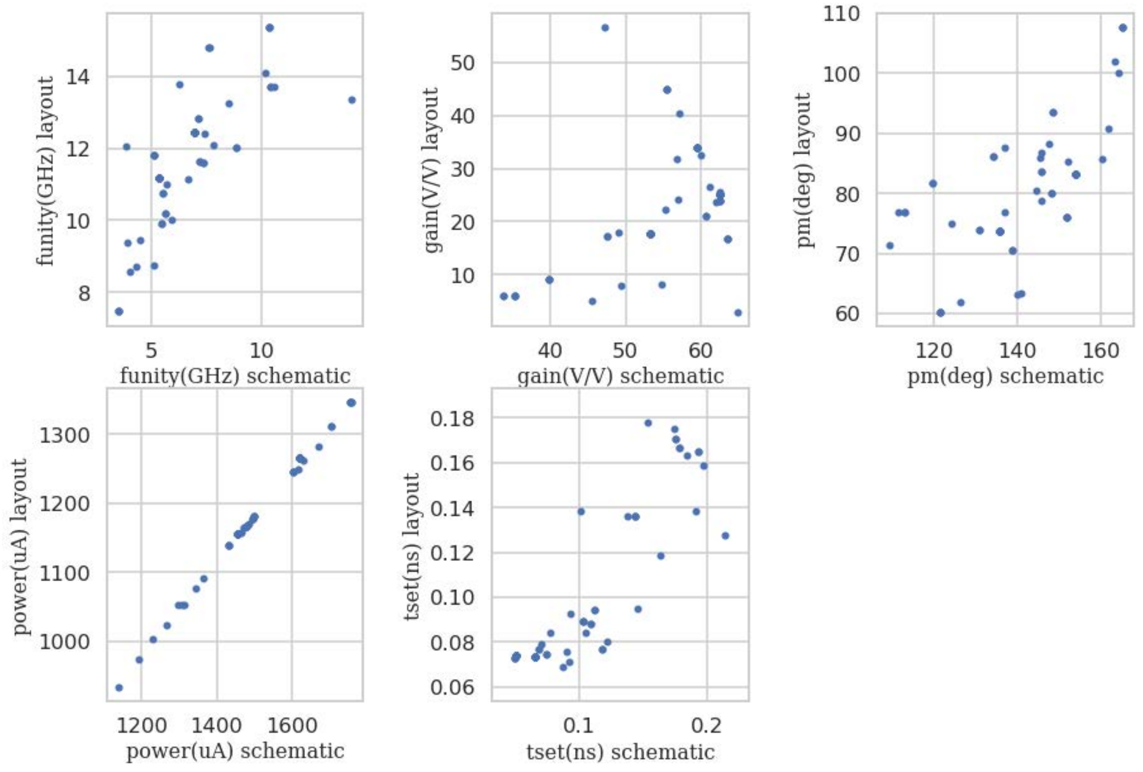


Figure 4.8: Scatterplot showing the difference between schematic and PEX simulations for each target metric for the ring amplifier.

To further assess this, we plot all of the metric performances the agent obtained, along with the expert design, in a series of two-dimensional metric-to-metric plots in Figure 4.12. This plot shows that the agent achieves designs that are sampled around the expert, shown in orange, with better and worse 2-D metric performances. In addition, the frontier curves in Figure 4.12 depict the theoretical performance boundaries that can be achieved by the agent. This information, in the future, can be used by designers to not only understand the limitations of the RL agent in design space, but also potentially understand the real and feasible boundaries for this circuit.

To better analyze the quality of performances the agent obtained, we compare an agent-obtained design to that of an expert for one target specification. The results are shown in Table 4.9, where we find that the agent creates, with the same technology and testbench setup, a design that reaches better metric performances than that of an expert in just 2 simulations.

These initial results are promising, as they demonstrate that this framework could potentially obtain new and better regions of operation in the design space, especially in circuit topologies whose sizing procedures are not well-defined. We emphasize that, however, that there are several steps to further the practicality of this algorithm for the RAMP: 1) a full closed-loop analysis should be conducted in order to establish the stability of the circuit with real system-level parasitic elements, and 2) process-corner variation must be considered during the sizing process to ensure that the agent-selected designs are feasible under real manufacturing constraints.

Table 4.7: Obtained layout simulation performance ranges by agent.

| Metric | Minimum | Mean | Maximum |
|---|---|---|---|
| $A_{vo}$ $(dB)$ | 5.48 | 34.2 | 55.9 |
| $f_t$ $(GHz)$ | 6.9 | 11.4 | 15.3 |
| $pm$ $(^{\circ})$ | 60.7 | 76.4 | 102.3 |
| $t_{set}$ $(ns)$ | 0.08 | 0.53 | 0.89 |
| $P$ $(\mu A)$ | 567 | 849 | 1284 |

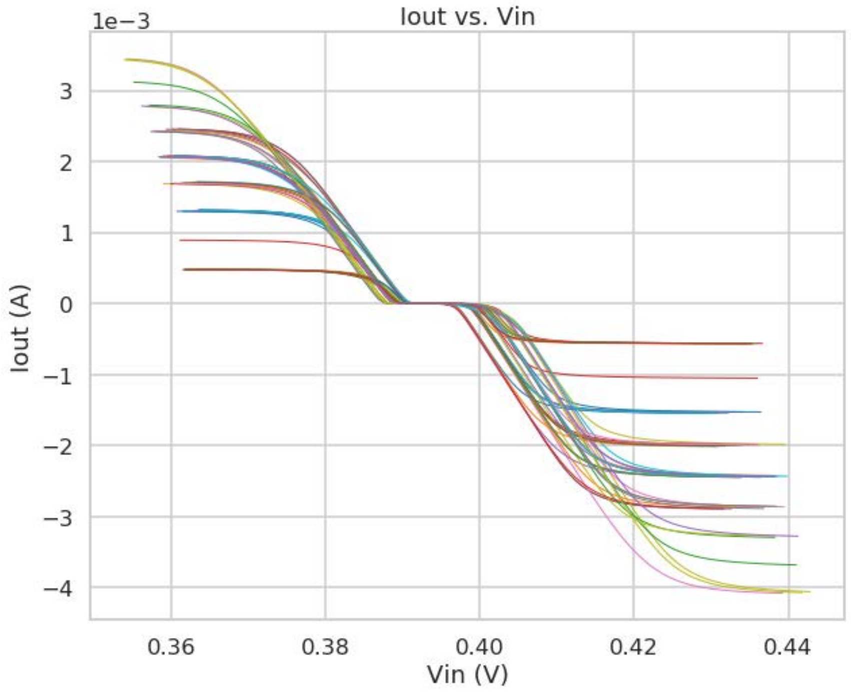


Figure 4.9: Output current versus input voltage layout simulation plot for the ring amplifier for different designs.

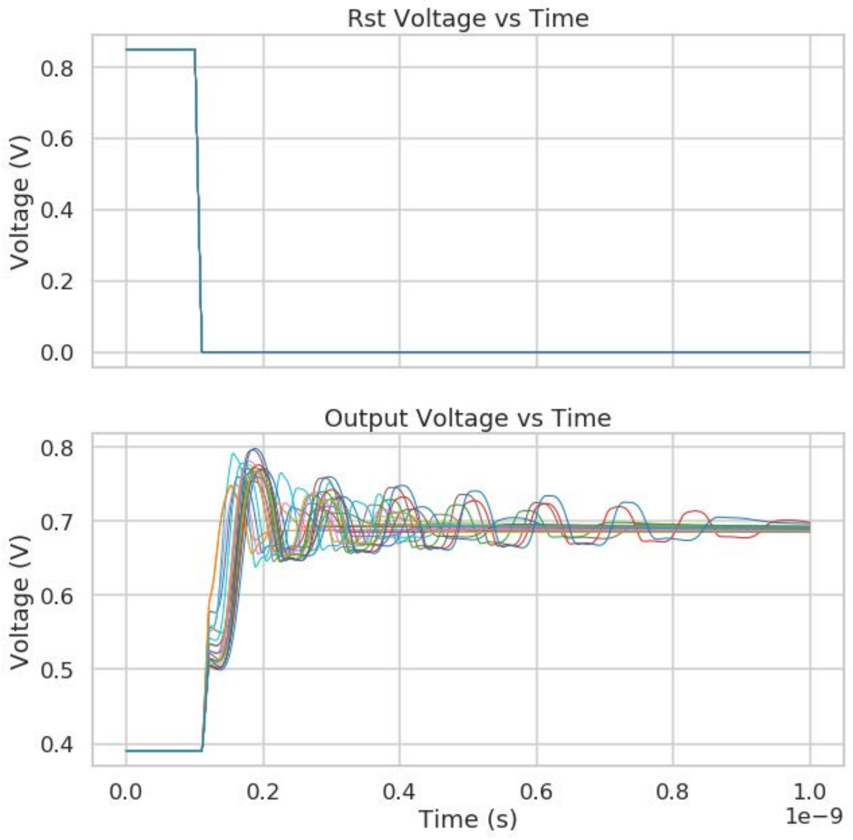


Figure 4.10: Ring amplifier layout simulation results for reset and output voltage versus time for different designs.

Table 4.8: Expert versus agent-obtained design for same target specification.

| Metric | Expert Design | Agent Design | Difference |
|---|---|---|---|
| $A_{vo}$ $(dB)$ | 27.83 | 39.36 | 41% higher |
| $f_t$ $(GHz)$ | 11.12 | 12.67 | 13.9% higher |
| $pm$ $(°)$ | 78.8 | 79.3 | 0.6% higher |
| $t_{set}$ $(ns)$ | 0.26 | 0.17 | 35% lower |
| $P$ $(\mu A)$ | 894 | 881 | 1.4% lower |

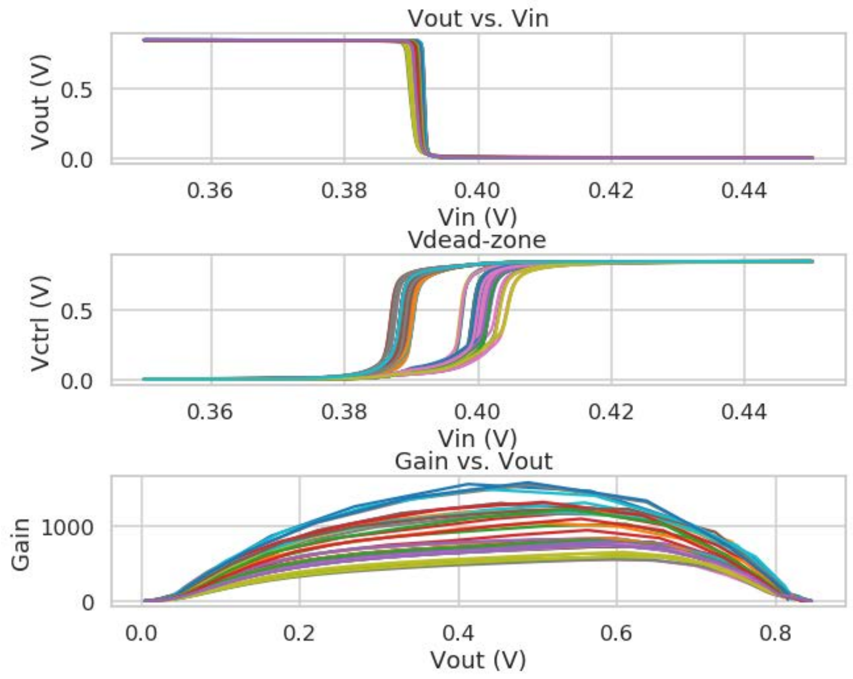


Figure 4.11: Obtained layout performance plots of output versus input voltage (top), dead-zone versus input voltage (middle), and gain versus output voltage (bottom) for the ring amplifier for different designs.

Table 4.9: Comparison of agent-obtained ring amplifier performance to an expert for the same target specification.

| Metric | Expert | Agent | Percent difference |
|---|---|---|---|
| $A_{vo}$ $(dB)$ | 27.8 | 39.4 | 41% |
| $f_t$ $(GHz)$ | 11.12 | 12.67 | 13.9% |
| $pm$ $(°)$ | 78.8 | 79.3 | 0.6% |
| $t_{set}$ $(ns)$ | 0.26 | 0.17 | 35% |
| $P$ $(\mu A)$ | 894 | 881 | 1.4% |

## 4.4 Conclusion

In this chapter, we tested the reinforcement-learning framework on two nonlinear circuit topologies: a strong-arm comparator and ring amplifier. We showed that the algorithm achieved many target specifications in a feasible range, taking on average just five simulations to converge to a schematic-simulated design that meets a target specification for the strong-arm comparator. In addition, expert-sized and agent-obtained designs are compared to

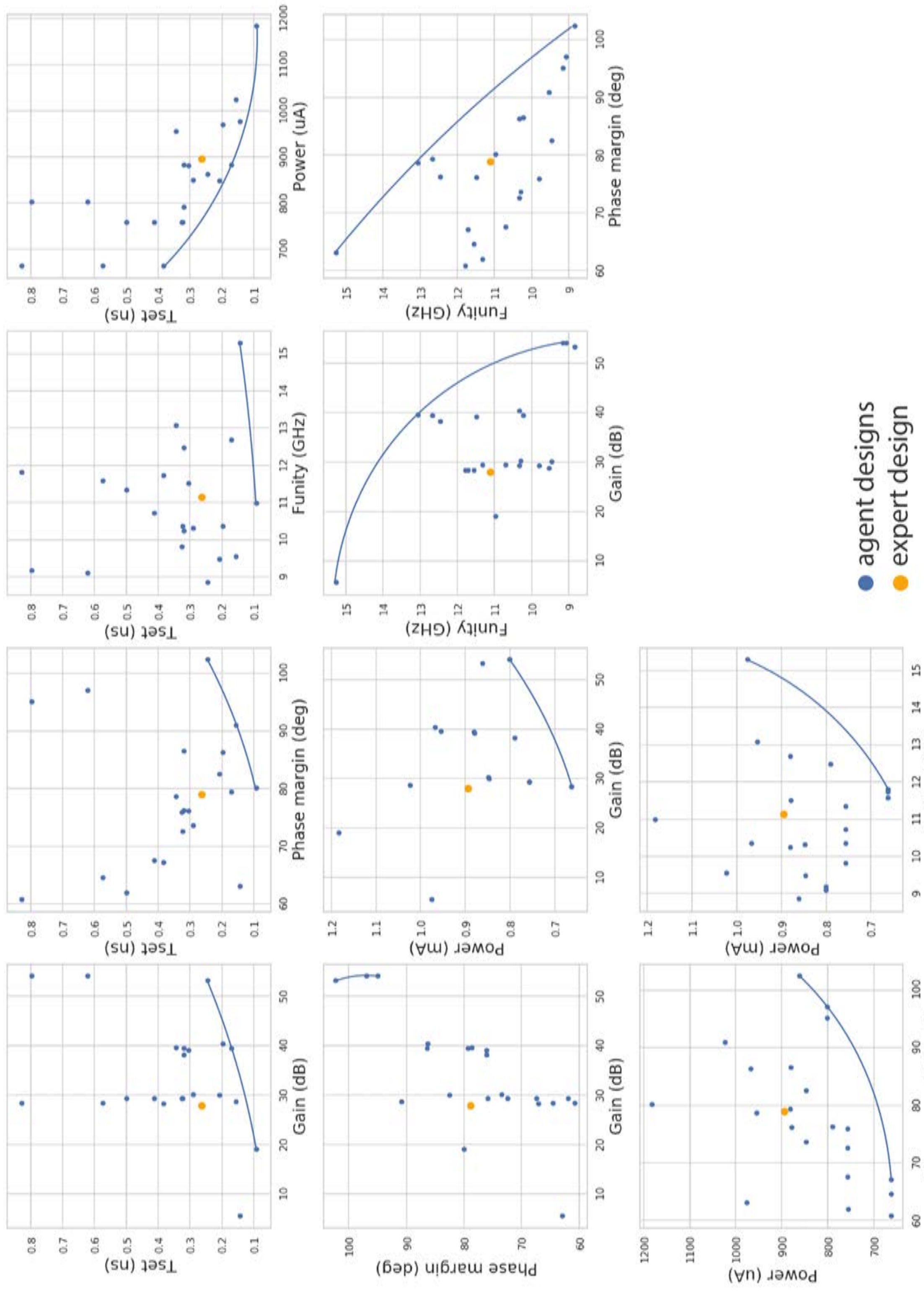


Figure 4.12: Pareto curves for reached performances by agent, along with the expert design.

understand both the practicality, feasibility, and value in this RL agent - and show that the agent achieves an average of 12% better performance than the expert.

We demonstrate similar results in a ring amplifier topology, where the results are validated in both schematic and layout simulations - to show that the agent obtains target specifications in two layout steps, while achieving a design that is 18% better than the expert. Methods are discussed to further improve and validate this framework, primarily by involving closed-loop simulations as well as process-corner variation. This is left as future work for this project.

# Chapter 5

# Analog Sub-Circuit Clustering with Graph Convolutional Neural Networks

We now apply machine learning in analog mixed-signal circuit design, particularly by clustering analog sub-circuits with graph convolutional neural networks (GCNN). We show that this approach entirely automates the clustering process while averaging 90% accuracy across six different analog circuits, ranging in size and complexity, taking just under one second to complete.

## 5.1 Analog synthesis and sub-circuit extraction background

In order to assess the efficacy of an automated analog synthesis tool that encapsulates design knowledge, it is important to consider several metrics specifically applicable to analog design automation:

- scalability dictates whether an algorithm successfully achieves its objective on a variety of different analog circuits, which is crucial for the practicality of any tool,
- accuracy measures how valid and correct the results of the algorithm are,
- degree of automation measures the level of involvement of the circuit designer. In an ideal scenario, this involvement is near zero,
- computational time calculates how long the algorithm takes to converge to a solution.

In this work, we present a framework to bridge the gap between schematic and layout generation by encapsulating the design intuition needed to create layout through the identification of critical analog sub-circuit structures. Prior work in this space can be split into three different categories: library-based, knowledge-based, and learning-based approaches.

Library-based methods involve the manual creation of many sub-circuit structures in the form of templates. Though they are accurate, these approaches lack robustness and automation, as the templates are often circuit-specific [75][76]. Knowledge-based methods codify rules to create constraints when generating clusters, and present the same pitfalls [77]. Learning-based approaches use supervision to learn sub-circuit structures; because data must be labelled, they require significant manual effort from an expert designer [78]. In addition, most make use of graph or hierarchical structures to represent a circuit topology, and often ignore additional features of the netlist that could aid in more accurate structure recognition.

We present a novel approach to the analog sub-circuit clustering problem by leveraging the ability of learning algorithms to recognize important features, while also generating the appropriate data needed to train this setup via an unsupervised clustering algorithm. We demonstrate that this methodology yields accurate results that work very well across different types of analog circuits with varying complexity, all while being entirely automated.

## 5.2 GCNN framework

Figure 5.1 shows the proposed framework, whose only input is a circuit netlist. The semi-supervised approach uses a graph convolutional neural network (GCNN) to cluster the devices in the topology based on feature data and a graph structure that represents net and device connections. The data used to train the GCNN is generated automatically, via the creation of partial labels and a graph netlist. These partial labels represent an initial guess about which cluster each device belongs to. In this section, we present the feature-extraction, cluster-count and label-extraction methods and as well as details of the graph implementation with the GCNN.

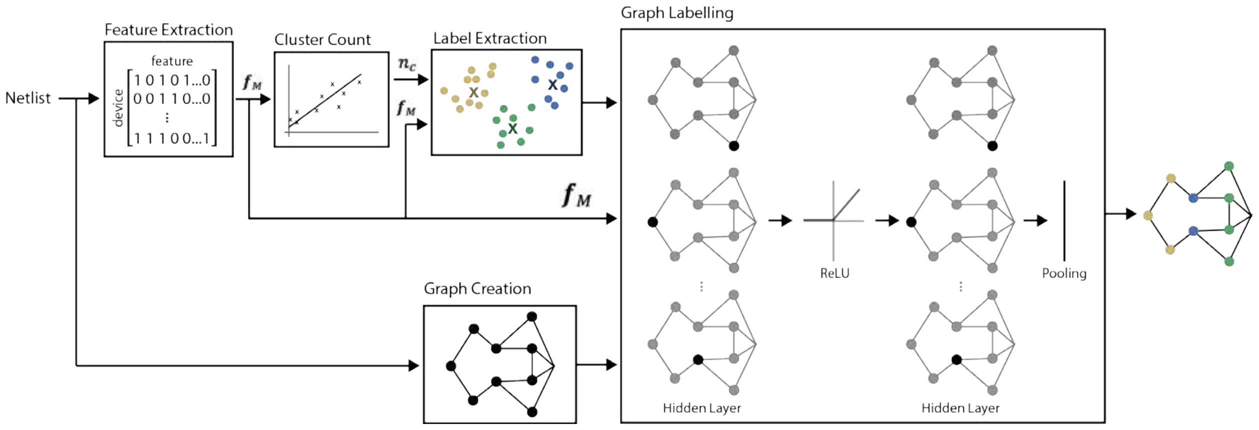


Figure 5.1: Total system overview of the analog clustering algorithm.

### 5.2.1 Feature extraction

The feature-extraction block automatically extracts the length, $x$ and $y$ coordinate of each device in the schematic, device orientation, device type and device weight, which is a measure of multiplicity and width, using Skill commands [79]. These features are one-hot encoded into a matrix $f_M$, and are given to the cluster count, label extraction and graph labelling blocks.

### 5.2.2 Cluster count and label extraction

Cluster counting and label extraction are used to automatically generate partial labels, which are then inputted to the graph labelling block. These labels are an initial guess as to which devices belong to a given cluster.

In particular, cluster counting consists of a linear model that takes as input total number of instances, number of N-type and P-type instances and number of nets to predict the number of total clusters, $n_c$, present in the circuit topology. This is then passed to the label extraction block, which uses a traditional k-means algorithm that takes both the feature matrix $f_M$ and $n_c$ as input, and returns the clustering results for each device in the netlist. This k-means algorithm on its own does not accurately cluster the devices, however, as it does not consider the net and device connections. In this framework, we use k-means to extract only one device label per cluster chosen from the converged centroid locations, where the device closest in distance to each centroid is used as the designated label.

### 5.2.3 Graph creation

A circuit netlist is programmatically converted to a graph, with each node representing a device and each edge representing the corresponding net connections. Traditional nMOS and pMOS devices have three edges representing the source, gate and drain connections. Bulk terminal connections are not included as they are connected similarly. Resistors and capacitors are two terminal devices represented by two edges and one node. In our studies, supply and grounds are included as a separate node.

### 5.2.4 Graph labelling

The graph labelling block consists of a GCNN that classifies all of the devices in a circuit netlist, from the initial partial labels provided by label extraction, $f_M$ and the graph netlist. In particular, we use a feature-based sub-graph clustering algorithm for semi-supervised node classification [80, 81]. The training procedure starts with the graph splitting into $k$ non-overlapping subgraphs, which are then used to estimate the parameters of the GCNN model based on the provided partial labels and feature information. The weights of the network are updated with stochastic gradient descent. In our studies, we fix the GCNN

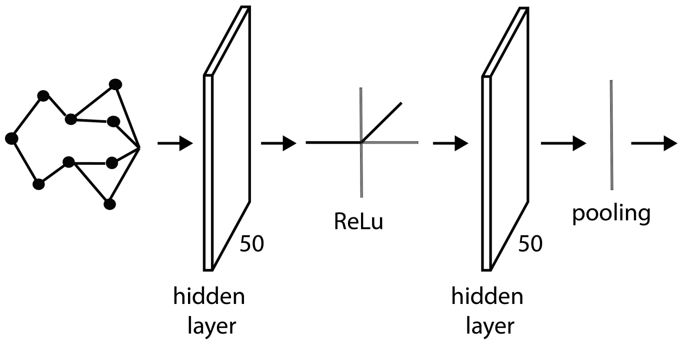


Figure 5.2: GCNN architecture showing two 50 neuron hidden layers, one ReLu non-linearity, and one pooling layer.

Table 5.1: Comparison table showing maximum number of devices in a tested topology, accuracy of algorithm, and automation.

| Algorithm | Max. # Devices | # of Circuits | Accurate? | Automated? |
|---|---|---|---|---|
| Library-based [76] | 22 | 2 | Y | No |
| Knowledge-based [77] | 37 | 5 | Y | No |
| ML-based [78] | 34 | 34 | N | No |
| This work | 35 | 6 | Y | Yes |

network architecture, shown in Figure 5.2 to be two layers, each with 50 neurons, and train the network until loss converges to within a preset $\epsilon$.

To determine the clustering accuracy, we compare the algorithm results to manually labelled circuit devices created apriori by an analog designer. The percent accuracy is determined on a device-by-device basis, wherein a device correctly belonging to the right cluster is accurate. These labels are used to compare the accuracy of the GCNN classifier during training, but are not required elsewhere in the algorithm.

## 5.3 Experimental setup and results

Table 5.2: Maximum accuracy of algorithm for multiple circuits.

| Circuit | Number of Devices | Number of Clusters | Accuracy |
|---|---|---|---|
| pll buffer | 4 | 2 | 100% |
| folded cascode | 35 | 9 | 91.3% |
| analog bias enable | 5 | 2 | 100% |
| current reference | 19 | 3 | 94.7% |
| opamp | 15 | 5 | 93.8% |
| comparator | 17 | 4 | 100% |

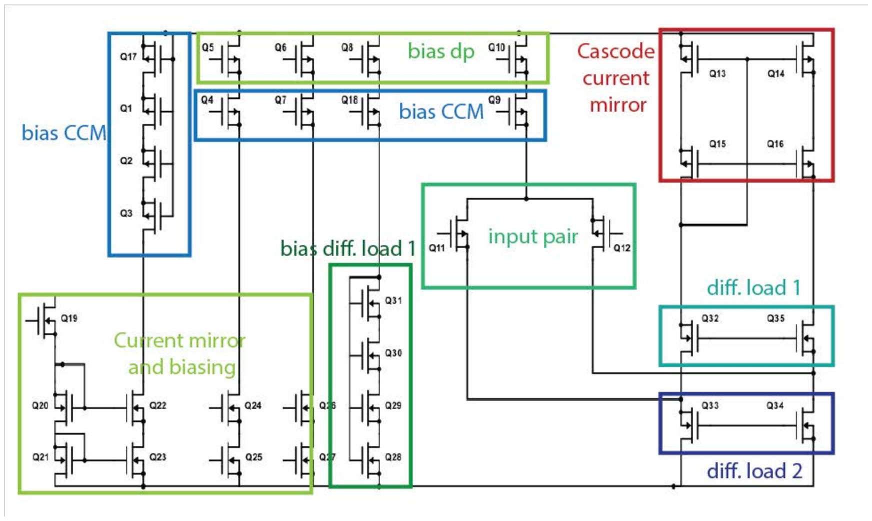


Figure 5.3: Folded cascode with biasing circuit, with hand-labelled clusters.

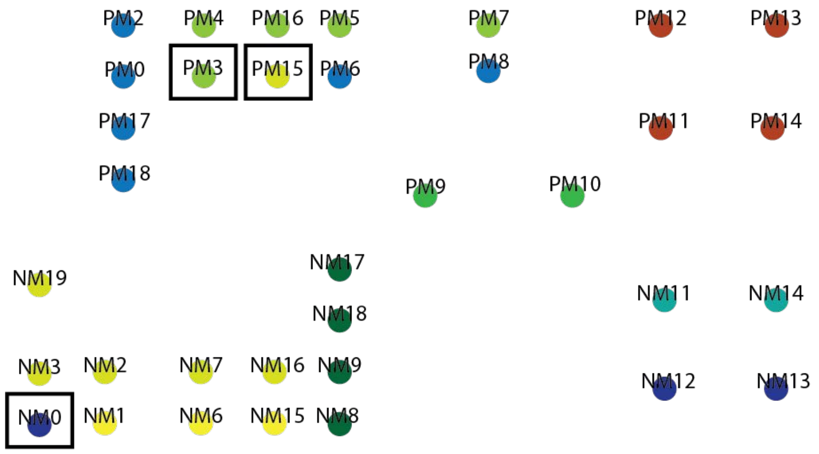


Figure 5.4: Output of GCNN algorithm visualized, the black boxes show incorrectly labelled devices, and the similarly-colored circles correspond to the same cluster.

We now test the framework on analog circuits designed with 16 nm technology models. Figure 5.3 shows a 35-transistor folded-cascode circuit topology with biasing, along with the expert-defined manual cluster assignments. For this topology, the clusters are assigned based on the functionality of each sub-circuit, particularly focusing on the identification of differential loads, biasing, input pairs, and current mirrors. Figure 5.4 shows the resulting node classification output of the GCNN, where the folded cascode obtains 91.3% accuracy. Note that the accuracy metric is calculated by comparing the correct cluster identifications with those obtained by the algorithm, device by device.

Table 5.1 shows how this algorithm compares with prior work. We test the algorithm on circuits with similar orders of complexity, and demonstrate that the framework is as robust as prior library and knowledge-based algorithms. Note that despite [78] tested on 34 circuits, their paper reports many inaccuracies in cluster assignments due to redundant, overlapping, or generic templates.

## 5.4 Analysis and conclusion

We now assess the algorithm in the context of important design automation metrics, which are crucial in determining the practicality and efficacy of this analog synthesis tool. We consider each separately below, and posit that this GCNN clustering methodology is an initial step to clustering analog circuits:

- This algorithm is robust, as we tested on six different circuits of varying complexity, size, and technology (see Table 5.2), with the GCNN architecture remaining the same for each topology.
- Higher than 90% accuracy is obtained for all tested circuits, which is important as this determines validity of the framework.
- The framework is automated, as the designer is not involved in any part of the clustering process process, other than providing an initial schematic topology. This, to the best of our knowledge, is one of the first tools to fully automate the analog sub-structure recognition task.
- Computationally, the algorithm takes a maximum of 1.13 seconds to complete for the most complex circuit, making it scalable.

In summary, the algorithm is the first to fully automate the analog sub-structure recognition task. We demonstrate that this framework is accurate and robust to multiple circuit topologies. Machine learning can be applied to non-traditional tasks, even ones that generally require human intervention and classification.

In the next chapter, we move to development and validation of a yield defect detection tool, to demonstrate that these same underlying benefits for machine learning are applicable in a different avenue of ASIC design.

# Chapter 6

# Yield Defect Detection using Convolutional Neural Networks

In this chapter, we demonstrate how machine learning can be applied to post-silicon verification, particularly with improving yield ramp-up time. We show that this approach:

- automates end-to-end layout defect pattern detection and extraction
- accurately finds layout defects once real diagnostic data is considered, and
- is orders of magnitude faster than manual approaches.

## 6.1 Defect-prone layout pattern detection

Yield is a critical factor in silicon technology scaling that determines cost per transistor [82, 83]. The time it takes to achieve a stable yield in new process technology nodes is increasing, however, primarily because of finer pitches and prominent design-process complexity. It is crucial, therefore, to improve this yield ramp-up rate so that profitability and quality objectives are met within a critical time window.

Typically, the introduction of a new technology involves improving the yield by detecting and fixing systematic defects, or non-random model-to-hardware defects, that occur due to process variations and aggressive design styles. Uncovering and mitigating these systematic defects [84] is not trivial given the complexity of present fault isolation (FI) and failure analysis (FA) techniques. Locating and analyzing defects involves various manual, expensive and time-consuming efforts, primarily to isolate, visualize, and determine the exact location and type of a defect that results in a functional failure. This often involves the expensive process of manually cutting and examining a die to assess physical failures.

Many works have addressed this problem by mining volume diagnosis data for identification of systematic defects, primarily by extracting patterns that best explain observed failures in large Integrated Circuit (IC) test datasets. These methods are useful because they do not need physical analysis, and instead only utilize available software test data.

Despite a variety of different test options for yield analysis, scan tests on a packaged part, and their associated diagnosis results are often used because they provide the necessary ability and observability to detect important defects that result in logical failures. Other tests include checking resistance and inspecting processing steps with images [85], but it is often difficult to determine if such observed abnormalities can result in a functional failure.

There are a number of prior works that automatically identify layout-dependent systematic defects from scan diagnosis tests with extra physical information such as design layouts [86, 87, 88, 89, 90]. Authors in [86] propose connecting layout-aware diagnosis to a Bayesian network model to learn the distribution of root-cause fault types with an expectation-maximization algorithm. Despite successfully testing their methodology on different yield detractors, this algorithm requires extensive feature extraction, which could be computationally expensive. Blanton, et.al. propose yield learning through non-test structures, and uses a combination of layout and diagnosis information, as well as image clustering to extract defective patterns [87]. This method, however, requires large amounts of pre-processing. In addition image clustering is not robust to noise variation in data. Keim, et.al. use scan diagnosis along with either process simulation, lithographic simulation, or design-for-manufacturing (DFM) rule application as failure rates to determine yield limiters. Their approach uses an iterative statistical procedure to extract the failure rate, but assumes an underlying diversity between yield limiting features [88]. Authors in [89] attempts to identify layout-dependent systematic defects by applying k-means clustering on layout snapshots in the form of images of suspects in volume diagnosis. K-means however, is susceptible to noise in data and relies on using an entire layout database to create clusters. Several works appeared as well to incorporate variability in the design of these circuits as well, through built-in tolerance [90]. These works are promising, but rely on correctly characterizing these variability in new technology nodes.

Despite the existence of these and other manual methods for extracting systematic defects [91, 92, 93, 94, 95, 96], identifying layout-sensitive systematic defects from scan diagnosis tests still remains quite challenging, primarily because of noise and complexity of layout patterns. In general, multiple heterogeneous defect modes are mixed with random defects in the diagnosis report, despite the diagnosis quality being high. Moreover, the number of layout patterns to explore scales exponentially with process technology, due to more layout layers. It therefore becomes extremely important to develop an algorithm to extract systematic defects that also scales with increasing complexity in process technology, as well as have the capability to handle larger amounts of noise.

### 6.1.1 Contribution

We propose a method to identify defect-prone layout patterns that manifest themselves as systematic defects from scan diagnosis and standard cell layouts by using a Convolutional Neural Network (CNN) to learn the layout features of these defects. CNNs are not only computationally efficient, but can also learn complex features while being more robust to noise, differentiating this method from other prior work that rely on image clustering techniques. In addition, because scan diagnosis and cell layouts are readily available, minimal pre-processing is required for their use to the CNN, while other works rely on extensive feature extraction from a large design layout database [87, 88, 89]. The following sections are organized as follows:

- Section 6.2 discusses how we train a CNN to extract layout features.
- Section 6.3 presents the how we extract particular layout defects using saliency detection, pattern extraction and ranking.
- Section 6.4 presents results of the methodology on both synthetic and real diagnostic data.
- Section 6.5 analyzes and discusses the approach in the context of yield learning, and the application of machine learning to circuit design.

## 6.2 Layout feature learning with convolutional neural networks

Deep convolutional neural networks are a well-known method in solving challenging computer vision tasks [97] such as object classification [98], localization [99], and object detection [100]. Specifically, a CNN's ability to generate an internal representation of a two or even three dimensional image allows it to extract visual features in an image, while being robust to noise variation in input data [97]. In this approach, we use CNNs to learn defect-prone layout features from noisy test data and complex cell layouts.

CNNs are trained in a supervised manner, requiring two pieces of data: 1) input images, and 2) the associated labels for these images. During training, the input images are passed through the CNN and the output is compared to the ground truth label, which is then back-propagated through each neuron in the network in the form of a gradient. Once training is complete, the weights of the trained CNN are frozen, and only forward passes are continued on new images.

We train the CNN on selected cell layouts, with their associated failure rate as labels. To determine the labels, we utilize scan diagnosis data, which analyzes scan test failures on a defective die and produces a list of suspects that potentially contain a real defect [91]. The diagnostic data is parsed as labels by calculating the failure rate of each cell by dividing the number of suspect cells per standard cell with the total number of instances of that cell in the design. If there are any patterns in the cell layouts that cause systematic defects, there will be elevations of failure rates for these specific cells.

The CNN will learn defect-prone layout patterns from cell layouts and scan diagnosis assuming the failure rate observed from the diagnosis data corresponds to the possibility of a standard cell containing a systematic defect.

There are several key points to validate this assumption:

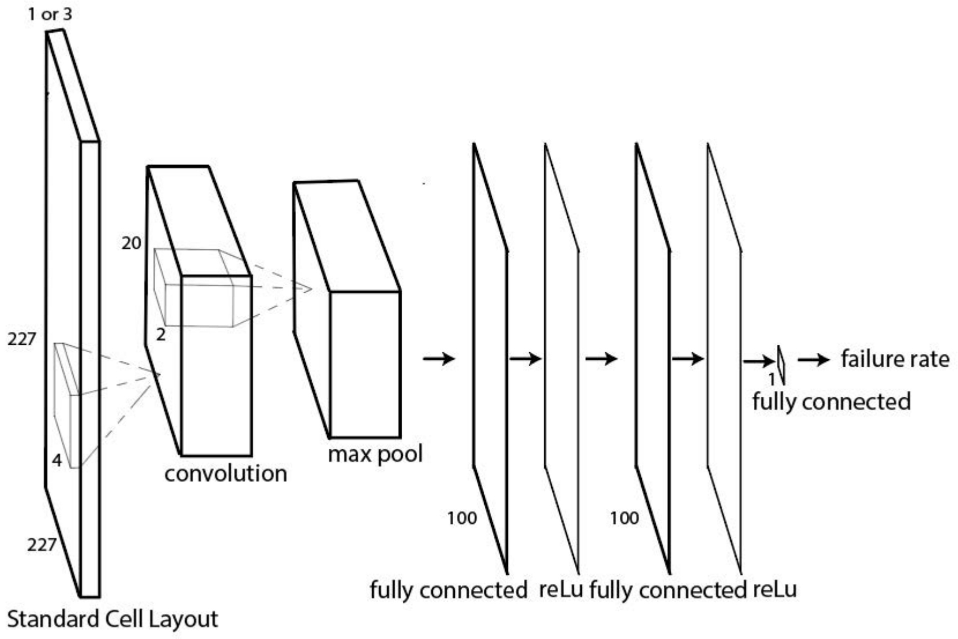


Figure 6.1: CNN architecture to learn defect-prone layout feature.

- Scan errors tend to be dominated by systematic defects, as evidenced by the the ratio of systematic defects to random defects significantly increasing in recent technology nodes, especially after the introduction of finFET.
- Front-end systematic defects, as opposed to back-end defects such as bridges and opens on routing metal layers, are predominant in the yield ramp-up process. Thus, the focus of these work is to improve the detection of front-end defects.
- Most systematic defects are layout-sensitive. It is known that the number of defects is highly correlated with specific local layout configurations.

Thus, the failure rate observed from the diagnosis data directly corresponds to the possibility of a standard cell containing a systematic defect, which assumes most of systematic defects are layout-sensitive.

The input images given to the CNN are standard cell layouts downsampled to 227x227 pixels. These layouts are obtained from an existing standard cell library, wherein one or more layers in the metal stackup, which are known to have a defective pattern, are used as the input. During downsampling, both the aspect ratio and scale are preserved to allow the CNN to learn size-sensitive layout defect patterns.

A CNN architecture with one convolutional layer, three fully connected layers, one pooling layer, and two rectified linear units is used (see Figure 6.1). The architecture is adapted from LeNet [97], a CNN that identifies digits. Further architecture modifications are made to optimize for network size by removing layers and reducing the kernel size of the convolutional filter to better suit the application to layout. In addition, we note that this network is relatively shallow and posit that this is due to solely identifying top-level layout patterns, requiring no need for additional feature detection found in deeper neural networks. The base learning rate is 0.00007 and RMSprop [101] is used as the gradient-based optimization method, for efficiency in determining the step size for the gradient.

The sigmoid cross-entropy loss function from Caffe [102] is used to estimate the failure rate labels by calculating the log loss from the difference between a true, labeled failure rate probability and the estimated failure probability from the output of the network.

Training is completed over 260 iterations, with a batch size of 20. These hyper-parameters are iterated to achieve the lowest loss and accuracy.

## 6.3 Layout pattern extraction and ranking

Once the CNN has been trained to a sufficient accuracy and loss, the saliency of the image is extracted, from which possible defect-prone layout patterns are highlighted. The salient images then pass through an algorithm that consolidates all patterns through the use of windowing and correlation.

### 6.3.1 Saliency extraction from learned CNN

Defect-prone layout patterns in a cell layout can be highlighted by using the trained CNN. [103] introduces a saliency-detection method wherein a trained CNN can be queried with the spatial support of a particular class in a given image. Intuitively, this implies that pixel-wise highlighting can be obtained with a given input image, its associated ground truth label, and the CNN-guessed output label.

Specifically, the non-linear layers of the CNN can be approximated by doing a first-order Taylor expansion [103]:

$$S_c(I) \approx w^T I + b \tag{6.1}$$

where

$$w = \delta S_c/\delta I|I_o \tag{6.2}$$

and $S_c(I)$ is the class score, $I_o$ is a specific image, $w$ are the network weights, and $b$ is the bias. The sensitivity of pixels in an image is calculated while freezing the weights in the trained CNN. This process involves just one back propagation for a given input image, from which the gradient is calculated. The pixels with the largest amount of gradient change affect the ground truth label the most.

Although the method in [103] was only applied to CNNs trained for classification, this work extends this saliency method to a regressive CNN with probability outputs, where the failure rate labels range from 0.0 to 1.0. The formulation predicts the failure rate of a given cell by using a normalized sigmoid later for density regression as opposed to the softmax

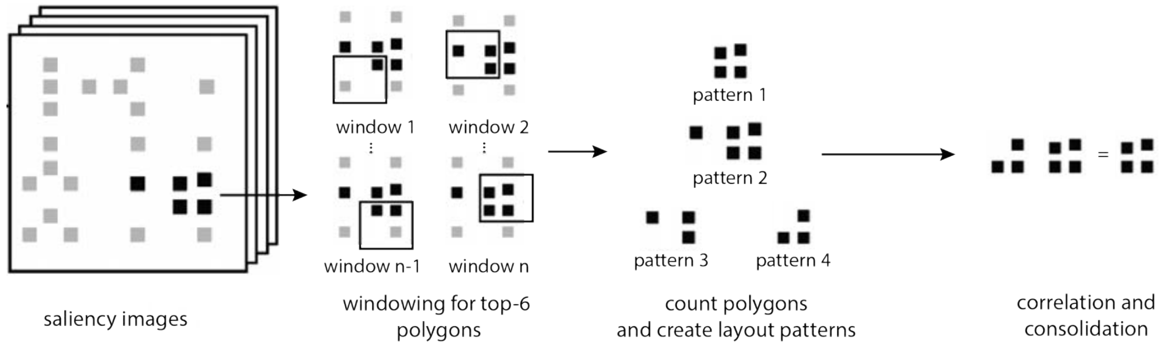

window 1
window 2
window n-1
window n
pattern 1
pattern 2
pattern 3
pattern 4
saliency images
windowing for top-6 polygons
count polygons and create layout patterns
correlation and consolidation

Figure 6.2: Extracting defect patterns (shown in black) with windowing pattern extraction.

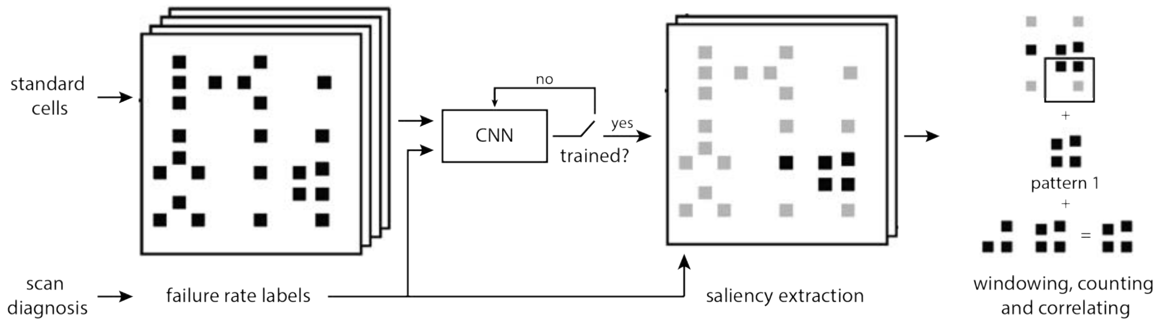


Figure 6.3: System level overview of defect-prone layout pattern-detection algorithm.

layer traditionally used for classification. The output is then fixed to the empirical failure rate of the cell and gradients are back-propagated. Mathematically, this can be thought of as changing the range of the class score $S_c(I)$ from a finite distribution to an infinite distribution, and calculating the gradient at this specific failure probability bias point given a cell layout. As Section 6.4 will show, saliency extraction can be successfully used in regression networks as well.

For each scan diagnosis report, the saliency image of every standard cell listed predicted as a failure is obtained. These images are then used as input to a pattern-extraction algorithm, presented in the next section.

### 6.3.2 Common defect pattern extraction from saliency map

The goal is to extract windows surrounding areas highlighted the most by saliency detection, and consolidate and match the resulting layout patterns. In the experiments, we demonstrate functionality on the via layer. The top-K highlighted regions within saliency are extracted by summing all of the pixel gradient values within a specific polygon appearing in the cell, where K is a parameter swept by the user. Given the tests on simulated and real diagnosis data, we found the optimal K value to be six.

For each top highlighted polygon, windows are drawn with window length $w$ and stride $s$. These parameters can be varied or swept, but are currently fixed for this framework. The layout patterns within these windows are extracted for each top-K highlighted rectangle within a standard cell, and are consolidated by the following steps, detailed in Figure 6.2:

- Scores are assigned to each via, based on how many times it appears in $w/s$ number of windows. Most systematic defects have polygon patterns that occur within close proximity to each other, which justifies taking this pre-processing step to reduce the number of cross-correlation computations that need to be done. In Figure 6.2, the vias highlighted in black correspond to the most salient polygons, and the scores assigned to them would be higher than the scores for the other polygons that do not appear as frequently in the windows.
- Layout patterns are then created by taking the top 3, 4 and 5 polygons for each standard cell top-K result. These patterns feature slices of the most often occurring layout patterns within each class of vias.
- Similarity scores are calculated across layout patterns with the same polygon count from which the top defect-prone layout patterns are extracted. Specifically, this score is obtained by using:

  $$sim\_score = \frac{I_A . * I_B}{max(area(I_A), area(I_B))}$$

  where $A$ and $B$ are two gray-scale images whose pixel values are either 0 or 1 for lack of, or present polygon, and where the area function is a tally of how much the layout is taken up by polygons. Orientation is also considered in this function, wherein rotational transformations are made to the input layout pattern. If the similarity score is higher than a pre-set threshold parameter, the two layout patterns are considered the same. This threshold parameter is set to account for possible variations in similar layout patterns having different distances between polygons.
- Cross-correlation is used across the extracted layout patterns with differing polygon counts, and are merged together if patterns are similar, using the same similarity metric. The resulting patterns are then used to obtain the ranking of the defect-prone patterns.

In addition to identifying defects with one layout layer as input, this method can be applied to multi-layer inputs to the CNN as well. The most salient layer is first obtained by summing all gradient values for each layer input, and then finding the maximum of the tabulated values. The step-by-step process presented above is then applied to this layer only. In addition, this methodology, with minimal modification, can also be used to identify multi-layer defects. Specifically, by changing the cross-correlation steps from two-dimensions to

Table 6.1: Accuracy, loss and divergence for the trained CNN.

| Accuracy | Loss | Kullback-Liebler Divergence |
|---|---|---|
| 98.2% | 0.213 | $0.2 - 0.64$ |

n-dimensions, defects present on multiple layers may be extracted by identifying the region in the cell with the combined most salient values.

The entire framework is shown in Figure 6.3. We note that the pattern extraction step, compared to prior work, only operates on likely defective layout cells, saving computational resources by not having to operate on the entire dataset.

## 6.4 Experimental results with synthetic and real diagnostic data

In this section we present this framework's performance on synthetically generated and real diagnostic data from a recent technology node.

### 6.4.1 Synthetic diagnostic data results

The input to the CNN is generated by injecting a synthetic defect in the form of a layout pattern into a subset of cells from a standard cell library. The failure rate associated with the defective cells varies with a normal distribution of 80% and a standard deviation of 15%. This variation is introduced to more realistically simulate lot-to-lot variation and noise from false suspects in scan diagnosis data. The CNN is given an even split of defective standard cells and non-defective standard cells.

Because of the introduction of random variation to the failure rate label, it is not correct to judge the CNN's accuracy on it's ability to correctly predict a ground-truth failure rate. For the purposes of validation, we create an accuracy metric used solely on the simulated dataset. In particular, if the CNN assigns a probability of greater than 0.5 to any defective cell or a probability of less than 0.5 to a non-defective cell, it is deemed correct. This assumption is reasonable given that the assigned failure probability of a defective cell has a mean greater than almost two standard deviations from 50% in this example. Using this metric, the calculated accuracy of the trained regression CNN was 98.2% (shown in Table 6.1).

The converged loss after training is 0.213. This loss value can be analyzed in context by deriving the minimum achievable entropy loss $H(p)$ where $p$ is the ground-truth failure probability:

$$H(p, q) = H(p) + D_{KL}(p||q)$$

where where $q$ is the estimated failure probability and $D_{KL}$ is is the Kullback-Leibler divergence of $q$ from $p$. The cross entropy is at its minimum value $H(p)$ when $p$ and $q$ are equal. The final loss, then, is in the range of $H(80 \pm 15\%)$, i.e. 0.2 to 0.64. This shows that

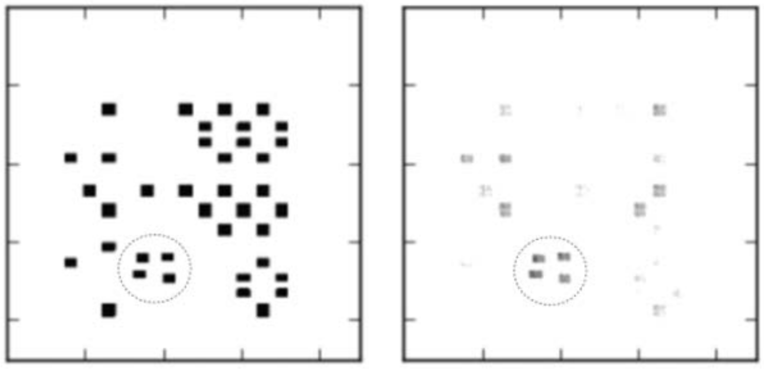


Figure 6.4: Cell layout (left) and its saliency image (right), with synthetic defect circled.

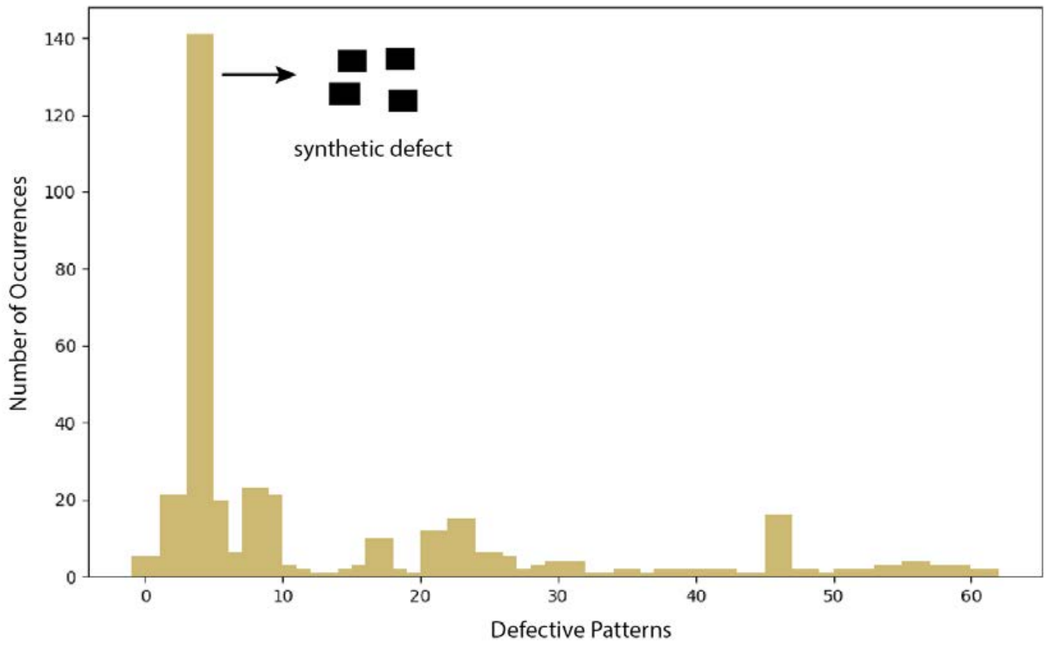


Figure 6.5: Results of applying windowing technique to saliency results.

the converged CNN loss value is just 6% above the lower bound, meaning that the CNN has optimized for estimating failure probability of cells. As the loss and accuracy values suggest, the CNN is able to differentiate between defect and non-defect prone standard cells accurately.

The results of implementing saliency detection on the trained convolutional neural network from Figure 6.4 shows the layout defect pattern highlighted clearly, with the four-via

Table 6.2: Test results on simulated diagnosis for different parameters.

| # Defects | # Unique Defects | # Layers | Defect Top-K Result | Layer Identified? |
|---|---|---|---|---|
| 1 | 1 | 1 | Top-1 | N/A |
| 1 | 1 | 3 | Top-1 | Yes |
| 1 | 2 | 3 | Top-2 in 4-via patt, Top-5 in 5-via patt | Yes |
| 2 | 2 | 3 | Top-1, Top-3 in 4-via patt | Yes |

defect having a larger concentration of dark, salient pixels. The pattern-detection algorithm is run on this trained regression CNN, and the output of the windowing results is plotted in Figure 6.5. The results show that the defect pattern appears in the created windows almost seven times more than the other windowing patterns, demonstrating that saliency extraction is able to detect a possible simulated synthetic defect pattern, which the framework can then successfully extract.

Three different parameter variation experiments are conducted on the synthetic dataset, summarized in Table 6.2. First, the number of defects maximally present in each standard cell unit is varied. The failure rate associated with the subset of cells with these injected defects is varied to reflect real diagnosis data. In particular, for each inserted defect, the mean failure rate is increased by 10%, and the standard deviation is decreased by 5%. Second, the number of unique defects is varied, to again reflect real diagnostic results that could have multiple layout failures. The failure rates associated with this experiment are also selected as before. Lastly, we vary the number of metal layers given to the agent, to show that this methodology can handle a realistic silicon stack. The injected defect pattern still occurs on one layer, with the pattern extraction algorithm changing such that the most salient layer is identified first, from which the extraction algorithm is carried out. In the future, we aim to test this methodology on multiple layer defects.

The results show that in all cases where defective patterns are injected into one layer, the method successfully identifies a) the defective layer and b) the correct failure pattern in the top-5 returned results. This not only demonstrates this framework's capability in identifying defects on synthetically generated diagnostic results, but also shows its robustness to noise. We note that despite the existence of random variability in failure rate, the defect extraction method is still successful.

### 6.4.2 Real diagnostic data results

We now test the effectiveness of the defective-layout pattern-detection methodology on product test diagnosis obtained from a recent technology node's yield ramp from Intel. The test data are three different diagnosis datasets from three unique lots, and indicate systematic defects that are highly correlated to a known defective-layout pattern.

The results from running the method on these diagnosis reports, summarized in Table 6.3

Table 6.3: Test results on real diagnosis data.

| Test Lot | Layer Identified? | # Defect Top-K Result |
|---|---|---|
| 1 | Yes | Top-1 |
| 2 | Yes | Top-3 |
| 3 | Yes | Top-4 |

show that it is able to identify not only defective layers but also the defect patterns within the top-5 extracted patterns from the windowing algorithm. We note a slight reduction in pattern extraction accuracy compared to the synthetic diagnostic test results, that we attribute to the presence of more noise in the real diagnosis reports, as well as multiple different defective patterns being present in the data.

The results show that despite the low failure rate of the standard cells and noise from false suspects in real diagnostic data, the regression CNN with saliency and pattern extraction is able to correctly identify the layout pattern causing a systematic defect. We compare these results to prior state-of-the-art in Table 6.4 and show that this work not only is less susceptible to noise, but requires less data and processing than prior work. In addition, we note that the results are demonstrated on both synthetic and unmodified real diagnostic reports, which is important for practicality.

## 6.5 Analysis and conclusion

In this chapter, we demonstrate that a defect-prone layout pattern can be automatically identified from scan diagnosis and standard library cell layouts using a CNN. This method first trains a CNN to learn yield-limiting layout features and then extracts and ranks the common layout patterns from the learned results. We show it successfully detects a defect-prone layout layer from all the simulated and real diagnosis data, and finds the problematic layout pattern from the layer within the top-5 defect-prone returned pattern list. In addition, compared to prior work, this algorithm is less susceptible to noise, and requires little data and pre-processing steps.

Practically, this algorithm reduces the layout search space significantly, narrowing it from hundreds of thousands to a manageable number of guessed defect-prone patterns. A yield engineer would now only have to analyze these small number of patterns to rule out possible false positives, and in the worst case would have to conduct physical failure analysis on all of the guessed defect-prone patterns returned by the algorithm. In addition, this framework can be used to also validate other methodologies, primarily as an initial pre-processing tool.

This work demonstrates that machine learning can be applied, understood and successfully validated in verification of post-silicon, particularly regarding yield ramp-up time. Importantly, it also reveals where machine-learning techniques are likely to be useful, primarily:

- in cases where large amounts of data must be processed in a computationally efficient manner,

- noise variation in data is considerable, preventing traditional methods from reliably finding a solution, and
- where underlying patterns, initially not known to a designer or are manually expensive to obtain, must be extracted and identified.

Table 6.4: Comparison table for prior yield defect detection works.

| Works | Required Data | Processing | # Lots Tested | Noise Prone | Real Defects |
|---|---|---|---|---|---|
| Bayesian network [86] | High | High | 4 | Medium | Yes |
| Physically-aware diagnosis [87] | High | High | - | High | Yes |
| Statistical clustering [89] | High | Medium | - | High | No |
| **Our Work** | Low | Low | 3 | Low | Yes |

# Chapter 7

# Conclusion

In this thesis, three avenues where machine learning can be applied to accelerate ASIC development are presented under the context of the six main requirements for automation in this domain.

## 7.1 Thesis contributions

In Chapter 2, an analog sizing framework that uses reinforcement learning to determine circuit parameters to meet a given target specification is introduced and tested on a transimpedance amplifier, vanilla two-stage amplifier, transimpedance amplifier with negative $g_m$ load, and a low dropout voltage regulator. Further analysis is conducted to assess the practicality and efficacy of the tool, showing that this methodology can accurately size these topologies in state-of-the-art run-time efficiency, while also being valid from a circuits perspective by additional comparisons to expert-sized designs.

Chapter 3 expands generalizability, interpretability, and practicability by testing on two parasitic-sensitive circuit topologies, a two-stage amplifier with negative $g_m$ load, and a two-stage folded-cascode with biasing. A new combined distribution deployment algorithm is used to improve run-time efficiency significantly, while additional analyses is used to ensure that the decisions the reinforcement learning agent makes are explainable and analyzable in the context of circuits.

Chapter 4 tests the RL framework on two nonlinear circuits: a strong-arm comparator and ring amplifier. These topologies require more design effort compared to other conventionally well-defined circuits; despite this, the RL agent remains successful in achieving target specifications, and obtains better performances when compared against a manually sized design from an expert, in both schematic and layout. Additional analyses is also conducted to show the pareto optimal boundary for the agent, providing potential for future applications to give feedback to the designer on a circuit's design space.

Finally, Chapters 5 and 6 present two projects where machine learning can be applied in other avenues of ASIC development. Specifically, Chapter 5 introduces a end-to-end, fully automated framework to extract clusters in analog sub-circuits using graph convolutional neural networks, obtaining high accuracy and run-time efficiency on six different circuits ranging in size and complexity. Chapter 6 presents a defect-prone layout pattern detection algorithm using convolutional neural networks and saliency extraction to identify defects in standard cell library layouts, and shows that this algorithm can consolidate the layout search space significantly by identifying defect patterns in an otherwise manual procedure.

## 7.2 Future directions

There are many ideas that can further the results and analyses presented in this thesis.

Specifically, the presented analog sizing framework in Chapter 2 does not take process corner/Monte Carlo process voltage and temperature simulations into account during the training process, which is the last step in making the design manufacturing-ready. This can potentially be incorporated by using the information from these simulation results in the same way as the layout simulation deployment methodology, presented in Chapter 3, and run process corner/Monte Carlo simulations only when the agent has obtained a valid schematic and layout performance design. If the results of these simulations do not meet a predefined acceptable variation, the worst performance is given back to the agent to once again go through the schematic and layout sizing procedure.

This thesis presents sizing results on seven unique circuit topologies of varying complexity, but it would be worth exploring how this algorithm could expand to consider system level design methodology, where a designer must map a high level system specification to specific targets in each analog sub-block. This problem is difficult, as this involves understanding the relationships and interactions between complex block-level analog circuits. Hierarchical reinforcement learning [104] is presented as a method to decompose a problem into a set of subtasks, so that each of the subtasks can be represented as reinforcement learning problems, and could be used in answering this system-level design issue.

In order to make a proposed automated solution practical in the context of the ASIC domain, the three projects presented in this thesis should be tested on more circuits and datasets to ensure generalizability to the variety of different information that can be given to these methods. This is especially true for the yield defect detection and analog sub-clustering algorithms from Chapters 5 and 6.

Finally, the algorithms presented in this thesis could be used in other avenues of ASIC development as well. For example, machine learning is already showing promise in digital place-and-route [105], but could also be used in power estimation, logic synthesis, and modelling simulators. If the analysis and chosen algorithms for these problems answer the six requirements, they have the ability of accelerating the ASIC development process significantly.

# Bibliography

[1] Monica Chhabra, Asavari Patil, and Supradip Baul. *ASIC Chip Market by Type (Full Custom, Semi- Based Custom, and Programmable Logic Devices) and Application (Aerospace Subsystem & Sensor, Wireless Communication, Medical Instrumentation, Telecommunication Products, Consumer Electronics, and Others): Globa.* Tech. rep. SE: Electronic Systems and Devices, 2019, p. 269. URL: https://www.alliedmarketresearch.com/asic-chip-market.

[2] Handel Jones. *FinFet and FD SOI: Market and Cost Analysis.* Tech. rep. Los Gatos: International Business Strategies, Inc., 2018. URL: https://soiconsortium.eu/wp-content/uploads/2018/08/MS-FDSOI9.1818-cr.pdf.

[3] David White et al. "MAGESTIC: Machine Learning Drive Automatic Generation of Electronic Systems Through Intelligent Collaboration". In: *Electron. Resurgence Initiat. Summit.* Defense Advanced Research Projects Agency, 2019.

[4] I. Bratko. "Machine Learning: Between Accuracy and Interpretability". In: *Learn. Networks Stat.* Vienna: Springer Vienna, 1997, pp. 163–177. DOI: 10.1007/978-3-7091-2668-4_10. URL: http://link.springer.com/10.1007/978-3-7091-2668-4%7B%5C_%7D10.

[5] *Cadence Design Systems, Inc., Virtuoso NeoCircuit.* Tech. rep. 2012, pp. 11–14.

[6] Dimitris Bertsimas and John Tsitsiklis. "Simulated annealing". In: *Stat. Sci.* (1993). ISSN: 08834237. DOI: 10.1214/ss/1177011077.

[7] Guyue Huang et al. "Machine Learning for Electronic Design Automation: A Survey". In: *Trans. Des. Autom. Electron. Syst.* (2021). arXiv: 2102.03357. URL: http://arxiv.org/abs/2102.03357.

[8] Luca Amarú, Pierre Emmanuel Gaillardon, and Giovanni De Micheli. "MIXSyn: An efficient logic synthesis methodology for mixed XOR-AND/OR dominated circuits". In: *Proc. Asia South Pacific Des. Autom. Conf. ASP-DAC.* 2013. ISBN: 9781467330299. DOI: 10.1109/ASPDAC.2013.6509585.

[9] Walter Lau Neto et al. "LSOracle: A logic synthesis framework driven by artificial intelligence: Invited paper". In: *IEEE/ACM Int. Conf. Comput. Des. Dig. Tech. Pap. ICCAD.* 2019. ISBN: 9781728123509. DOI: 10.1109/ICCAD45719.2019.8942145.

[10] Maryam Parsa et al. "Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design". In: *Front. Neurosci.* (2020). ISSN: 1662453X. DOI: `10.3389/fnins.2020.00667`.

[11] Prabir C. Maulik, L. Richard Carley, and Rob A. Rutenbar. "Integer programming based topology selection of cell-level analog circuits". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* (1995). ISSN: 02780070. DOI: `10.1109/43.372366`.

[12] Andreas Gerlach et al. "A generic topology selection method for analog circuits with embedded circuit sizing demonstrated on the OTA example". In: *Proc. 2017 Des. Autom. Test Eur. DATE 2017*. 2017. ISBN: 9783981537093. DOI: `10.23919/DATE.2017.7927115`.

[13] James T.O. Neill. *An overview of neural network compression.* 2020. arXiv: `2006.03669`.

[14] Stephen P. Boyd et al. "Digital circuit optimization via geometric programming". In: *Oper. Res.* (2005). ISSN: 0030364X. DOI: `10.1287/opre.1050.0254`.

[15] Paolo Cicconi et al. "A constraint-based approach for optimizing the design of overhead lines". In: *Int. J. Interact. Des. Manuf.* (2020). ISSN: 19552505. DOI: `10.1007/s12008-020-00680-x`.

[16] Kourosh Hakhamaneshi et al. "BagNet: Berkeley Analog Generator with Layout Optimizer Boosted with Deep Neural Networks". In: *2019 IEEE/ACM Int. Conf. Comput. Des.* IEEE, 2019, pp. 1–8. ISBN: 978-1-7281-2350-9. DOI: `10.1109/ICCAD45719.2019.8942062`. arXiv: `1907.10515`. URL: `https://ieeexplore.ieee.org/document/8942062/`.

[17] Miri Weiss Cohen, Michael Aga, and Tomer Weinberg. "Genetic Algorithm Software System for Analog Circuit Design". In: *Procedia CIRP* 36 (2015), pp. 17–22. ISSN: 22128271. DOI: `10.1016/j.procir.2015.01.033`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S2212827115000360`.

[18] Amod P Vaze. "Analog Circuit Design using Genetic Algorithm : Modified". In: *World Acad. Sci. Eng. Technol.* (2006).

[19] Keertana Settaluri et al. "AutoCkt: Deep Reinforcement Learning of Analog Circuit Designs". In: *Proc. 2020 Des. Autom. Test Eur. Conf. Exhib. DATE 2020.* 2020. ISBN: 9783981926347. DOI: `10.23919/DATE48585.2020.9116200`. arXiv: `2001.01808`.

[20] Amjad Mohsen and Richard Hofmann. "Power modeling, estimation, and optimization for automated co-design of real-time embedded systems". In: *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* (2004). ISSN: 16113349. DOI: `10.1007/978-3-540-30205-6_66`.

[21] Robert Kirby et al. "CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks". In: *IEEE/IFIP Int. Conf. VLSI Syst. VLSI-SoC.* 2019. ISBN: 9781728139159. DOI: `10.1109/VLSI-SoC.2019.8920342`.

[22] Yi-Chen Lu et al. "GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization". In: *2019 IEEE/ACM Int. Conf. Comput. Des.* IEEE, 2019, pp. 1–8. ISBN: 978-1-7281-2350-9. DOI: 10.1109/ICCAD45719.2019.8942063. URL: https://ieeexplore.ieee.org/document/8942063/.

[23] Haoxing Ren et al. "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks". In: *Proc. - Des. Autom. Conf.* 2020. ISBN: 9781450367257. DOI: 10.1109/DAC18072.2020.9218515.

[24] J. Crossley et al. "BAG: A designer-oriented integrated framework for the development of AMS circuit generators". In: *IEEE/ACM Int. Conf. Comput. Des. Dig. Tech. Pap. ICCAD.* 2013. ISBN: 9781479910717. DOI: 10.1109/ICCAD.2013.6691100.

[25] Edward Wang et al. "A Methodology for Reusable Physical Design". In: *Proc. - Int. Symp. Qual. Electron. Des. ISQED.* 2020. ISBN: 9781728142074. DOI: 10.1109/ISQED48828.2020.9136999.

[26] Andreas Olofsson. "Intelligent Design of Electronic Assets (IDEA) & Posh Open Source Hardware (POSH)". In: *DARPA/MTO.* 2018.

[27] Dave Reed. *Analog Design Needs to Change.* Tech. rep. Semiconductor Engineering, 2020. URL: https://semiengineering.com/analog-design-needs-to-change/.

[28] Nuttorn Jangkrajarng et al. "IPRAIL—intellectual property reuse-based analog IC layout automation". In: *Integration* 36.4 (2003), pp. 237–262. ISSN: 01679260. DOI: 10.1016/j.vlsi.2003.08.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167926003000701.

[29] Paul G. A. Jespers and Boris Murmann. *Systematic Design of Analog CMOS Circuits.* Cambridge: Cambridge University Press, 2017. ISBN: 9781108125840. DOI: 10.1017/9781108125840. URL: http://ebooks.cambridge.org/ref/id/CBO9781108125840.

[30] E. Berkcan and M. D'Abreu. "Physical assembly for analog compilation of high voltage ICs". In: *Proc. IEEE 1988 Cust. Integr. Circuits Conf.* IEEE, 1988, pp. 14.3/1–14.3/7. DOI: 10.1109/CICC.1988.20865. URL: http://ieeexplore.ieee.org/document/20865/.

[31] Kaustubha Mendhurwar et al. "A new approach to sizing analog CMOS building blocks using pre-compiled neural network models". In: *Analog Integr. Circuits Signal Process.* 70.3 (2012), pp. 265–281. ISSN: 0925-1030. DOI: 10.1007/s10470-011-9648-z. URL: http://link.springer.com/10.1007/s10470-011-9648-z.

[32] Walter Daems, Georges Gielen, and Willy Sansen. "An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits". In: *Proc. 2002 Des. Autom. Conf. (IEEE Cat. No.02CH37324).* New Orleans: IEEE, 2002, pp. 431–436. ISBN: 1-58113-461-4. DOI: 10.1109/DAC.2002.1012664. URL: http://ieeexplore.ieee.org/document/1012664/.

[33] Wenlong Lyu et al. "An Efficient Bayesian Optimization Approach for Automated Optimization of Analog Circuits". In: *IEEE Trans. Circuits Syst. I Regul. Pap.* 65.6 (2018), pp. 1954–1967. ISSN: 1549-8328. DOI: `10.1109/TCSI.2017.2768826`. URL: `https://ieeexplore.ieee.org/document/8116661/`.

[34] Prerit Terway, Kenza Hamidouche, and Niraj K. Jha. "DISPATCH: Design Space Exploration of Cyber-Physical Systems". In: *arXiv Prepr. arXiv2009.10214* (2020). ISSN: 23318422. arXiv: `2009.10214`. URL: `http://arxiv.org/abs/2009.10214`.

[35] Glenn Wolfe and Ranga Vemuri. "Extraction and use of neural network models in automated synthesis of operational amplifiers". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 22.2 (2003), pp. 198–212. ISSN: 0278-0070. DOI: `10.1109/TCAD.2002.806600`. URL: `http://ieeexplore.ieee.org/document/1174095/`.

[36] Yaping Li et al. "An Artificial Neural Network Assisted Optimization System for Analog Design Space Exploration". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 39.10 (2020), pp. 2640–2653. ISSN: 0278-0070. DOI: `10.1109/TCAD.2019.2961322`. URL: `https://ieeexplore.ieee.org/document/8937720/`.

[37] Wook Lee and Frans A. Oliehoek. "Analog Circuit Design with Dyna-Style Reinforcement Learning". In: *arXiv Prepr. arXiv2011.07665* (2020). ISSN: 23318422. arXiv: `2011.07665`. URL: `http://arxiv.org/abs/2011.07665`.

[38] Hanrui Wang et al. "Learning to Design Circuits". In: *arXiv Prepr. arXiv1812.02734* (2018). arXiv: `1812.02734`. URL: `http://arxiv.org/abs/1812.02734`.

[39] Kai-En Yang et al. "Fast Design Space Adaptation with Deep Reinforcement Learning for Analog Circuit Sizing". In: *arXiv Prepr. arXiv2009.13772* (2020). URL: `https://arxiv.org/pdf/2009.13772.pdf`.

[40] David J. Chen and Bing J. Sheu. "Automatic layout synthesis of analog ICs using circuit recognition and constraint analysis techniques". In: *Analog Integr. Circuits Signal Process.* 1.1 (1991), pp. 75–87. ISSN: 0925-1030. DOI: `10.1007/BF02151027`. URL: `http://link.springer.com/10.1007/BF02151027`.

[41] Nuno Lourenço, Ricardo Martins, and Nuno Horta. "Layout-Aware Sizing of Analog ICs using Floorplan & Routing Estimates for Parasitic Extraction". In: *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2015.* New Jersey: IEEE Conference Publications, 2015, pp. 1156–1161. ISBN: 9783981537048. DOI: `10.7873/DATE.2015.0411`. URL: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7092562`.

[42] Husni Habal and Helmut Graeb. "Constraint-Based Layout-Driven Sizing of Analog Circuits". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 30.8 (2011), pp. 1089–1102. ISSN: 0278-0070. DOI: `10.1109/TCAD.2011.2158732`. URL: `http://ieeexplore.ieee.org/document/5956867/`.

[43] I-Lun Tseng, Adam Postula, and Lech Jozwiak. "Symbolic Extraction for Estimating Analog Layout Parasitics in Layout-Aware Synthesis". In: *IEEE Mix.* Citeseer, 2005, pp. 195–199.

[44] Oleg Garitselov, Saraju P. Mohanty, and Elias Kougianos. "A comparative study of metamodels for fast and accurate simulation of nano-CMOS circuits". In: *IEEE Trans. Semicond. Manuf.* (2012).

[45] Saraju P. Mohanty and Elias Kougianos. "Incorporating manufacturing process variation awareness in fast design optimization of nanoscale CMOS VCOs". In: *IEEE Trans. Semicond. Manuf.* (2014).

[46] Dhruva Ghai, Saraju P. Mohanty, and Elias Kougianos. "Design of parasitic and process-variation aware Nano-CMOS RF circuits: A VCO case study". In: *IEEE Trans. Very Large Scale Integr. Syst.* (2009).

[47] Rafael Castro-Lopez et al. "An Integrated Layout-Synthesis Approach for Analog ICs". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 27.7 (2008), pp. 1179–1189. ISSN: 0278-0070. DOI: `10.1109/TCAD.2008.923417`. URL: `http://ieeexplore.ieee.org/document/4544854/`.

[48] Pat Langley. *Elements of Machine Learning.* San Francisco: Morgan Kaufmann Publishers, Inc., 1996, pp. 1–10. ISBN: 1-55860-301-8.

[49] *Amazon Machine Learning Developer Guide.* Tech. rep. Amazon Web Services, Inc., 2020, pp. 1–4. URL: `https://docs.aws.amazon.com/machine-learning/latest/dg/machinelearning-dg.pdf%7B%5C#%7Dwhen-to-use-machine-learning`.

[50] Qianwen Wang et al. "ATMSeer: Increasing Transparency and Controllability in Automated Machine Learning". In: *Proc. 2019 CHI Conf. Hum. Factors Comput. Syst.* New York, NY, USA: ACM, 2019, pp. 1–12. ISBN: 9781450359702. DOI: `10.1145/3290605.3300911`. arXiv: `1902.05009`. URL: `https://dl.acm.org/doi/10.1145/3290605.3300911`.

[51] John Schulman et al. *Proximal Policy Optimization Algorithms.* Tech. rep. OpenAI, 2017.

[52] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *35th Int. Conf. Mach. Learn. ICML 2018* (2017). arXiv: `1712.09381`. URL: `http://arxiv.org/abs/1712.09381`.

[53] Eric Chang et al. "BAG2: A process-portable framework for generator-based AMS circuit design". In: *2018 IEEE Cust. Integr. Circuits Conf.* IEEE, 2018, pp. 1–8. ISBN: 978-1-5386-2483-8. DOI: `10.1109/CICC.2018.8357061`. URL: `https://ieeexplore.ieee.org/document/8357061/`.

[54] *Virtuoso Visualization and Analysis XL User Guide.* Tech. rep. Cadence, 2012.

[55] Zhaokai Liu, Borivoje Nikolic, and Vladimir Stojanović. "Time-interleaved SAR ADC Design Using Berkeley Analog Generator". MA thesis. University of California, Berkeley, 2020. URL: `https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-109.pdf`.

[56] Willy Sansen. “Optimum Opamp design in one day”. In: *Symp. VLSI Technol. Circuits.* 2021.

[57] Behzad Razavi. “The Low Dropout Voltage Regulator”. In: *IEEE Solid-State Circuits Mag.* (2019).

[58] Samiran DasGupta and Pradip Mandal. “An automated design approach for CMOS LDO regulators”. In: *2009 Asia South Pacific Des. Autom. Conf.* IEEE, 2009, pp. 510–515. ISBN: 978-1-4244-2748-2. DOI: 10.1109/ASPDAC.2009.4796531. URL: http://ieeexplore.ieee.org/document/4796531/.

[59] Yen-Lung Chen et al. “Simultaneous optimization for low dropout regulator and its error amplifier with process variation”. In: *Tech. Pap. 2014 Int. Symp. VLSI Des. Autom. Test.* IEEE, 2014, pp. 1–4. ISBN: 978-1-4799-2776-0. DOI: 10.1109/VLSI-DAT.2014.6834870. URL: http://ieeexplore.ieee.org/document/6834870/.

[60] Dan Mercer. *University Courses Chapter 20: Analog to Digital Conversion.* Tech. rep. Analog Devices, 2021. URL: https://wiki.analog.com/university/courses/electronics/text/chapter-20.

[61] N. Bako, Ž Butković, and A. Barić. “Design of fully differential folded cascode operational amplifier by the Gm / ID methodology”. In: *MIPRO 2010 - 33rd Int. Conv. Inf. Commun. Technol. Electron. Microelectron. Proc.* 2010. ISBN: 9789532330502.

[62] Meysam Akbari and Omid Hashemipour. “High gain and high CMRR two-stage folded cascode OTA with nested miller compensation”. In: *J. Circuits, Syst. Comput.* (2015). ISSN: 02181266. DOI: 10.1142/S0218126615500577.

[63] Darrell Whitley. “A genetic algorithm tutorial”. In: *Stat. Comput.* 4.2 (1994). ISSN: 0960-3174. DOI: 10.1007/BF00175354. URL: http://link.springer.com/10.1007/BF00175354.

[64] Huan Wang et al. “On the generalization gap in reparameterizable reinforcement learning”. In: *36th Int. Conf. Mach. Learn. ICML 2019.* 2019. ISBN: 9781510886988. arXiv: 1905.12654.

[65] Izel Cagin Odabasi et al. “A Rare Event Based Yield Estimation Methodology for Analog Circuits”. In: *2018 IEEE 21st Int. Symp. Des. Diagnostics Electron. Circuits Syst.* IEEE, 2018, pp. 33–38. ISBN: 978-1-5386-5754-6. DOI: 10.1109/DDECS.2018.00013. URL: https://ieeexplore.ieee.org/document/8410502/.

[66] Mitali Ahuja, Soumya Narang, and Satwik Patnaik. “A process corner detection methodology for resilience towards process variations using adaptive body bias”. In: *IEEE Int. Conf. Circuit, Power Comput. Technol. ICCPCT 2015.* 2015. ISBN: 9781479970759. DOI: 10.1109/ICCPCT.2015.7159326.

[67] Behzad Razavi. “The Design of a Comparator”. In: *IEEE Solid-State Circuits Mag.* (2020). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9265306.

[68] Behzad Razavi. “The Design of a Comparator [The Analog Mind]”. In: *IEEE Solid-State Circuits Mag.* 12.4 (2020), pp. 8–14. ISSN: 1943-0582. DOI: 10.1109/MSSC.2020.3021865. URL: https://ieeexplore.ieee.org/document/9265306/.

[69] Abdullah Almansouri et al. “Improved StrongARM latch comparator: Design, analysis and performance evaluation”. In: *PRIME 2017 - 13th Conf. PhD Res. Microelectron. Electron. Proc.* 2017. ISBN: 9781509065073. DOI: 10.1109/PRIME.2017.7974114.

[70] Benjamin Hershberg and Un-Ku Moon. “The Ring Amplifier: Scalable Amplification with Ring Oscillators”. In: *High-Performance AD DA Convert. IC Des. Scaled Technol. Time-Domain Signal Process.* Cham: Springer International Publishing, 2015, pp. 399–418. DOI: 10.1007/978-3-319-07938-7_18. URL: http://link.springer.com/10.1007/978-3-319-07938-7%7B%5C_%7D18.

[71] Jorge Lagos et al. “A 1-GS/s, 12-b, Single-Channel Pipelined ADC With Dead-Zone-Degenerated Ring Amplifiers”. In: *IEEE J. Solid-State Circuits* 54.3 (2019), pp. 646–658. ISSN: 0018-9200. DOI: 10.1109/JSSC.2018.2889680. URL: https://ieeexplore.ieee.org/document/8618444/.

[72] Behzad Razavi. “The StrongARM latch [A Circuit for All Seasons]”. In: *IEEE Solid-State Circuits Mag.* (2015). ISSN: 19430582. DOI: 10.1109/MSSC.2015.2418155.

[73] Jorge Lagos et al. “A single-channel, 600-MS/s, 12-b, ringamp-based pipelined ADC in 28-nm CMOS”. In: *IEEE J. Solid-State Circuits* (2019). ISSN: 00189200. DOI: 10.1109/JSSC.2018.2879923.

[74] Joschua Conrad et al. “Design Approach for Ring Amplifiers”. In: *IEEE Trans. Circuits Syst. I Regul. Pap.* (2020). ISSN: 15580806. DOI: 10.1109/TCSI.2020.2986553.

[75] Tobias Massier, Helmut Graeb, and Ulf Schlichtmann. “The sizing rules method for CMOS and bipolar analog integrated circuit synthesis”. In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* 2008. DOI: 10.1109/TCAD.2008.2006143.

[76] Markus Meissner and Lars Hedrich. “FEATS: Framework for explorative analog topology synthesis”. In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* (2015). ISSN: 02780070. DOI: 10.1109/TCAD.2014.2376987.

[77] Po Hsun Wu et al. “A novel analog physical synthesis methodology integrating existent design expertise”. In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* (2015). ISSN: 02780070. DOI: 10.1109/TCAD.2014.2379630.

[78] Hao Li, Fanshu Jiao, and Alex Doboli. “Analog circuit topological feature extraction with unsupervised learning of new sub-structures”. In: *Proc. 2016 Des. Autom. Test Eur. Conf. Exhib. DATE 2016.* 2016. ISBN: 9783981537062. DOI: 10.3850/9783981537079_0923.

[79] Sobhana Tayenjam, Venkata Narayana Rao Vanukuru, and S. Kumaravel. "A PCell design methodology for automatic layout generation of spiral inductor using skill script". In: *2017 Int. Conf. Microelectron. Devices, Circuits Syst. ICMDCS 2017*. 2017. ISBN: 9781538617168. DOI: 10.1109/ICMDCS.2017.8211572.

[80] Wei Lin Chiang et al. "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks". In: *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 2019. ISBN: 9781450362016. DOI: 10.1145/3292500.3330925. arXiv: 1905.07953.

[81] Thomas N. Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.* 2017. arXiv: 1609.02907.

[82] William M. Holt. "Moore's law: A path going forward". In: *Dig. Tech. Pap. - IEEE Int. Solid-State Circuits Conf.* 2016. ISBN: 9781467394666. DOI: 10.1109/ISSCC.2016.7417888.

[83] Kenneth Flamm. "Has Moore's Law Been Repealed? An Economist's Perspective". In: *Comput. Sci. Eng.* (2017). ISSN: 15219615. DOI: 10.1109/MCSE.2017.30.

[84] Liang Teck Pang et al. "Measurement and analysis of variability in 45 nm strained-Si CMOS technology". In: *IEEE J. Solid-State Circuits* (2009). ISSN: 00189200. DOI: 10.1109/JSSC.2009.2022217.

[85] Israel Koren and Zahava Koren. "Defect tolerance in VLSI circuits: Techniques and yield analysis". In: *Proc. IEEE* (1998). ISSN: 00189219. DOI: 10.1109/5.705525.

[86] Brady Benware et al. "Determining a failure root cause distribution from a population of layout-aware scan diagnosis results". In: *IEEE Des. Test Comput.* (2012). ISSN: 07407475. DOI: 10.1109/MDT.2011.2178386.

[87] Ronald De Shawn Blanton et al. "Yield learning through physically aware diagnosis of IC-failure populations". In: *IEEE Des. Test Comput.* (2012). ISSN: 07407475. DOI: 10.1109/MDT.2011.2178587.

[88] Martin Keim et al. "A rapid yield learning flow based on production integrated layout-aware diagnosis". In: *Proc. - Int. Test Conf.* 2006. ISBN: 1424402921. DOI: 10.1109/TEST.2006.297715.

[89] Wing Chiu Jason Tam and Ronald D.Shawn Blanton. "LASIC: Layout Analysis for Systematic IC-Defect Identification Using Clustering". In: *IEEE Trans. Comput. Des. Integr. Circuits Syst.* (2015). ISSN: 02780070. DOI: 10.1109/TCAD.2015.2406854.

[90] Zheng Guo et al. "Large-scale SRAM variability characterization in 45 nm CMOS". In: *IEEE J. Solid-State Circuits* (2009). ISSN: 00189200. DOI: 10.1109/JSSC.2009.2032698.

[91] Leo Breiman. "Statistical modeling: The two cultures". In: *Stat. Sci.* (2001). ISSN: 08834237. DOI: 10.1214/ss/1009213726.

[92] Leendert M. Huisman, Maroun Kassab, and Leah Pastel. "Data mining Integrated Circuit fails with fail commonalities". In: *Proc. - Int. Test Conf.* 2004. DOI: `10.1109/test.2004.1387327`.

[93] Jay Jahangiri and David Abercrombie. "Value-added defect testing techniques". In: *IEEE Des. Test Comput.* (2005). ISSN: 07407475. DOI: `10.1109/MDT.2005.74`.

[94] W. Maly and J. Deszczka. "Yield estimation model for VLSI artwork evaluation". In: *Electron. Lett.* (1983). ISSN: 00135194. DOI: `10.1049/el:19830156`.

[95] Manish Sharma et al. "Efficiently performing yield enhancements by identifying dominant physical root cause from test fail data". In: *Proc. - Int. Test Conf.* 2008. ISBN: 9781424424030. DOI: `10.1109/TEST.2008.4700589`.

[96] Ritesh Turakhia et al. "Bridging DFM analysis and volume diagnostics for yield learning - A case study". In: *Proc. IEEE VLSI Test Symp.* 2009. ISBN: 9780769535982. DOI: `10.1109/VTS.2009.37`.

[97] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proc. IEEE* (1998). ISSN: 00189219. DOI: `10.1109/5.726791`.

[98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* (2017). ISSN: 15577317. DOI: `10.1145/3065386`.

[99] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *2nd Int. Conf. Learn. Represent. ICLR 2014 - Work. Track Proc.* 2014. arXiv: `1312.6034`.

[100] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.* 2014. ISBN: 9781479951178. DOI: `10.1109/CVPR.2014.81`. arXiv: `1311.2524`.

[101] Sebastian Ruder. "Overview Optimization Gradients". In: *arXiv Preprint* (2017). ISSN: 0006341X. arXiv: `1609.04747`.

[102] Yangqing Jia et al. "Caffe: Convolutional architecture for fast feature embedding". In: *MM 2014 - Proc. 2014 ACM Conf. Multimed.* 2014. ISBN: 9781450330633. DOI: `10.1145/2647868.2654889`. arXiv: `1408.5093`.

[103] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.* 2015.

[104] Yanhua Huang. "Hierarchical reinforcement learning". In: *Deep Reinf. Learn. Fundam. Res. Appl.* 2020. DOI: `10.1007/978-981-15-4095-0_10`.

[105] Azalia Mirhoseini et al. "A graph placement methodology for fast chip design". In: *Nature* (2021). ISSN: 14764687. DOI: `10.1038/s41586-021-03544-w`.