

REINFORCEMENT LEARNING FOR SEQUENTIAL DECISION-MAKING:
FROM CHIP DESIGN TO LANGUAGE MODELING

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Anna Goldie
June 2025

© 2025 by Anna Darling Goldie. All Rights Reserved.
Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-3.0 United States License.

<http://creativecommons.org/licenses/by/3.0/us/>

This dissertation is online at: <https://purl.stanford.edu/kc333cx8703>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chelsea Finn

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Azalia Mirhoseini

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

Reinforcement learning (RL) methods have achieved impressive results in simulated environments and games, but only recently have they been deployed to solve real-world problems. This is due in part to the instability of these methods, their relative sample inefficiency, and the difficulty of attributing reward to individual steps within the long trajectories characteristic of real-world tasks. To address these challenges, this dissertation presents approaches to problem formulation, representation learning, effective reward attribution, and scalable curation of high-quality multi-step trajectories. To ground these principles, I will introduce RL agents capable of solving real-world challenges in two disparate areas: chip design and language modeling. First, I will describe AlphaChip, a deep reinforcement learning method capable of generating superhuman chip layouts in hours, rather than weeks or months of human effort. AlphaChip was one of the first reinforcement learning methods deployed to solve a real-world engineering problem, and it has designed chip layouts in the last four generations of Google TPU, as well as other chips across Alphabet and by external chipmakers. Next, I will introduce Step-Wise Reinforcement Learning (SWiRL), a reinforcement learning and synthetic data generation method that improves the ability of large language models (LLMs) to perform multi-step reasoning and use tools. Finally, I conclude with a new dataset for evaluating the performance of LLM-based RL agents on challenging multi-step reasoning tasks, and discuss open problems and opportunities in this new frontier.

Acknowledgments

I would like to start by thanking my advisor, Christopher Manning. When I was considering PhD programs, my absolute top choice was to be at Stanford and to be advised by Chris. I grew up reading Chris’s NLP textbooks, and I have such incredible respect for him, for the depth and breadth of his research contributions (I have heard him referred to as “the pope of NLP”, and I feel this is no exaggeration), for his commitment to advising his students, for the clarity and enthusiasm with which he teaches NLP concepts. My esteem for him has only grown as we worked together longer, and in different capacities (as a research assistant, and as head TA for his course). I appreciate that he was always careful to give me freedom and autonomy, even when my research interests diverged from those he normally pursued. If not for his open-mindedness in this regard, I would never have been able to write the dissertation that follows. I also appreciate all of his incredibly detailed feedback on this dissertation. It made me feel like a true part of the linguistics lineage that I carry through him, and I enjoyed the opportunity to collaborate deeply on one last piece of writing together. Chris is a legend, and I have grown so much from having him as my advisor.

I would also like to thank my advisor, Azalia Mirhoseini. She is an absolutely brilliant researcher, a true visionary, and a deeply funny and kind person as well. We first met during the Google Brain Residency in 2016, and we have worked together ever since. Joining Google Brain in 2016 was the best career decision I have ever made; this was true for many reasons, but chief among them was that I met Azalia. Our partnership has been invaluable to me, and the greatest work of my life was done in collaboration with her. We’ve experienced both great joy and intense pain in our research journey together, and I am so grateful that neither of us had to face such experiences alone. Although in some sense it felt odd to take on an advisor-student relationship after having worked as equal partners for many years, it is a great honor to be her first student, and I am glad that we will share this additional bond throughout life. I hope that we will always remain close collaborators, as I feel that anything is possible when we work together.

I am very grateful to Quoc Le for supporting me in my dream of earning a PhD, and for serving on my oral committee. I feel extremely lucky to have gotten to work with Quoc and to have had him as a manager and mentor for so many years. He is brilliant, creative, curious, inspiring, and one of the most impressive researchers I have ever met, and yet somehow he is also funny, humble, and

down-to-earth. As a manager, he has given me the freedom and autonomy to pursue my greatest work, and has been there for me through the most challenging times.

I greatly appreciate Chelsea Finn for serving on both my reading and oral committee. She is one of my RL heroes, so I felt incredibly honored when she agreed to serve on my committee. I am very grateful that she took the time to engage with my work and ask interesting questions at my defense!

I am also very grateful to Amin Saberi for serving as external chair of my committee. The moment I met him, I was struck by his intelligence and quick wit. When it came time to select my external chair, he was my first and immediate choice. He executed his duties flawlessly, managing the discussion and asking deep and wide-ranging questions himself.

I am especially thankful to Bill Dally, who shaped the course of my life when I met him at MLSys back in 2018. When he heard that I was going to do my PhD at MIT in the fall, he immediately said that I should come to Stanford instead and work with Chris Manning, although I had not applied to the program that year due to my own error. I have no idea how he could so quickly and confidently come to this conclusion, but he was right. He is one of the most impressive individuals I have ever met, and his brain is truly overclocked with seemingly instantaneous reaction speeds. In any case, I am extremely grateful to him for his advice and support, which have always served me well.

I'd like to sincerely thank Dave Patterson for his technical advice, for his deep wisdom about life and career, and for his support throughout my PhD and my time at Google.

I would like to thank Jeff Dean for his deep support over many years, for collaborating with me on exciting research projects, for the joy of our technical discussions, for convincing me to join Google Brain back in 2016, and for agreeing to let me do my PhD while maintaining my position at Google. This dissertation would never have been possible without this unique combination of academia and industry. Jeff is someone I greatly admire, and if I had to pick one person that I'd want to be more like when I "grow up", I would have to say Jeff. He is just an incredible combination of brilliance, strategic thinking, mental velocity, technical depth, integrity, empathy, kindness, and punny humor. My favorite part of working at Google is having lunch with him and Azalia, and talking about technical challenges at the intersection of systems and machine learning.

I'd like to thank all of my amazing coauthors across all of the papers in this dissertation, and beyond it. None of this would have been possible without them, and I learned so much from working with incredible people from Google Brain, Stanford, and Anthropic throughout my PhD.

I'd like to thank the people who patiently gave me advice when I was agonizing over whether to do a PhD, including those who vehemently advised me not to! These generous individuals include Ion Stoica, Hugo Larochelle, Chris Olah, Greg Brockman, Sergey Levine, Yoshua Bengio, Pieter Abbeel, Dan Jurafsky, Percy Liang, Jim Glass, Stephanie Seneff, Song Han, Joshua Tenenbaum, Michael Jordan, Richard Socher, Luke Zettlemoyer, among many others.

I am grateful to the students in the Stanford CS department who always made me feel welcome.

I especially want to thank the members of the NLP group and of SAIL more broadly, as well as Chris's and Azalia's students, including Alex Tamkin, Shikhar Murty, Moussa Doumbouya, Katelyn Zhou, Kaylee Burns, Nelson Liu, John Hewitt, Ethan Chi, Parth Sarthi, Jasper Jian, Ryan Chi, Jordan Jurafsky, Jon Saad-Falcon, Anne Ouyang, Caia Costello, Dilara Soylu, Robert Csordas, Mo Tiwari, Martijn Bartelds, Tolulope Ogunremi, Houjun Liu, Derek Chong, Julie Kallini, Zhengxuan Wu, Xinran Zhao, among many others.

I'd like to thank Google and its leadership for supporting me in my research career and in earning this PhD. I am especially grateful to Jeff Dean, Zoubin Ghahramani, Ed Chi, Samy Bengio, Koray Kavukcuoglu, Demis Hassabis, Doug Eck, D. Sculley, Anna Patterson, and everyone else who supported me during difficult times and who offered me amazing opportunities.

I'd like to thank Stephanie Seneff, my research advisor at MIT. Working with her was my first experience doing research, and I am so grateful that she took me under her wing. I am also grateful to Victor Zue for giving the lecture on dialogue systems that inspired me to go into this field back in 2004, as well as the Spoken Language Systems Group more broadly. I am grateful also to the Women's Technology Program at MIT that provided me with my first taste of electrical engineering, computer science, and discrete math.

I'd like to thank Anne Hunter for her help in dealing with an incredibly challenging situation at MIT that no one else was able to. Her assistance fundamentally altered the course of my life, and removing this obstacle allowed me to pursue an MEng and conduct independent research for the first time. This opened doors for me at Google Research and Google Brain, and ultimately made my career what it is today. I'd also like to thank my friend Jenny Liu for referring me to Google back in 2013.

I'd like to thank my high school science team coach and earth science teacher, Dr. Steve Wilkins. He believed in me endlessly and well before he had any reason to. He strongly encouraged me to pursue a career in research and engineering, and he put great effort into helping me grow. For example, I was so nervous when giving presentations that he had me come in early every Wednesday morning to give a practice talk. Although I didn't feel like I was improving at the time, I am so grateful to him now.

I'd like to thank my aunt(-in-law) Sarah Freedman for her advice and support over many years. When I first moved to the Bay Area to work for Google, she allowed me to stay at her house for months while I searched for housing. I am continuously inspired by her intelligence, energy, generosity, and lifelong love for learning and life. No one in my immediate family had ever gone into academia or gotten a PhD, so it was especially helpful to get the advice and perspective of a Berkeley faculty member as I navigated my PhD.

I also am especially thankful for my mother and father-in-law, who have shown me true love and care, and have celebrated the wins together and comforted me during challenging times.

I am grateful to my friends Katya Dreyer-Oren and Jessa Dickinson for being there for me since we were 4 years old, and for all the amazing friends I've met in the intervening years, including Aviv Ovadya, Edwin Chen, Mark Avara, Seastar Lin, Maithra Raghu, Jacob Steinhardt, Greg Belote, Katy Blumer, Victoria Dean, Hugh Oh, Tyler Lu, Leonid Shamis, Omari Stephens, Naomi Saphra, Simon Safar, Jelani Nelson, Chris Baranowski, Martin Froehlich, Mo Bavarian, Sharon Zhou, Rosemary Ke, Been Kim, Pia Pal, Advay Mengle, Aakanksha Chowdhery, and many others.

I'd like to thank my family for supporting me always: my Mom and Dad, my siblings William and Sam, and my grandmother who is turning 100 in just 6 days. I've always felt accepted and loved just as I am, and free to pursue my dreams and passions without judgment. Being born into this family is the greatest luck of my life, and I cannot imagine a kinder, sweeter, more intelligent, or engaging tribe to have gotten to grow up around and go through life beside.

Finally, I'd like to thank my husband Gabriel Warshauer-Baker. I feel incredibly lucky to have met him at MIT, though I suppose he would say that he didn't leave that up to chance. He is brilliant, handsome, funny, and kind. It is often said that the most important decision in life is one's choice of a life partner, and I have found that to be undoubtedly true. There's no way I could have done this PhD without Gabe's incredible support and kindness during many difficult times. He has been a truly selfless partner, and has always done everything he could to make sure that I was happy and could achieve my dreams, and he is the reason that all of them have come true. He is the light of my life and the person I look forward to seeing at the end of every day.

Contents

Abstract	iv
Acknowledgments	v
1 Introduction	2
1.1 Reinforcement Learning for Sequential Decision-Making	2
1.2 A Brief History of Reinforcement Learning	3
1.3 Thesis Statement, Research Questions, and Contributions	5
1.4 Overview of Dissertation	7
1.5 Bibliographic Notes	8
I Reinforcement Learning for Chip Design	13
2 Placement Optimization With Reinforcement Learning	14
2.1 Background	14
2.2 Placement Optimization as an RL Problem	14
2.3 Principles of Deep Reinforcement Learning	15
2.4 Deep Reinforcement Learning for Placement Optimization	17
2.4.1 Placement Problem Formulation	17
2.4.2 Graph Convolutional Neural Networks	19
2.4.3 Domain Adaptation	19
2.4.4 Solving the Placement Objective with Policy Gradient Optimization	20
2.5 Ingredients for RL Success	20
2.6 Conclusion	22
3 AlphaChip: A Graph Placement Method for Chip Design	23
3.1 Background	23
3.2 Introduction	24
3.3 Related Work	26

3.4	Chip floorplanning as a learning problem	28
3.5	Designing Domain-Adaptive Policies	29
3.6	Methods	32
3.6.1	Problem Statement	32
3.6.2	Overview of Our Approach	32
3.6.3	Detailed Methodology	33
3.6.4	Synthesis of the input netlist	34
3.6.5	Selection of grid rows and columns	34
3.6.6	Selection of macro order	34
3.6.7	Clustering of Standard Cells	35
3.6.8	Generation of Adjacency Matrix	35
3.6.9	Placement of Standard Cells	35
3.6.10	Postprocessing	36
3.6.11	Reward	36
3.6.12	Action Representation	37
3.6.13	State Representation	37
3.6.14	Enabling Transfer Learning	37
3.6.15	Policy Network Architecture	39
3.6.16	Policy Network Update: Training Parameters	40
3.7	Empirical Evaluation	40
3.7.1	Experimental Setup	40
3.7.2	Open-Source Benchmark: Ariane RISC-V	41
3.7.3	Google TPU Results: Comparing with Baseline Methods	43
3.7.4	Domain Adaptation Results	45
3.7.5	Learning from Larger Datasets	46
3.8	Discussion	46
3.8.1	Use in a Production Setting	46
3.8.2	Impact of Cost Trade-offs	47
3.8.3	Robustness to Noise	48
3.8.4	Generalization vs. Training Data	48
3.8.5	Insights and Visualizations	49
3.8.6	Implications for a Broader Class of Problems	50
3.9	Conclusion	51
	Appendices	52
3.A	Early Exploration of RL Optimization Algorithms	52
3.B	Exploring RL Convergence Properties	54

3.C	Exploring Effect of Input Ordering	55
3.D	Post-RouteOpt Validation of AlphaChip	56
3.E	Comparing with Simulated Annealing	58
4	From Scrutiny to Silicon: AlphaChip Post-Publication	60
4.1	Background	60
4.2	Errors in Attempted Reproduction of Our Method	64
4.2.1	No Pre-Training Performed for RL Method	64
4.2.2	RL Method Provided with Far Fewer Compute Resources	65
4.2.3	RL Method Not Trained to Convergence	66
4.2.4	Test Cases Not Representative of Modern Chips	68
4.3	Transparency & Reproducibility	68
4.3.1	AlphaChip is Fully Open-Source	68
4.3.2	Claims They Cannot Share Their “Open” Test Cases	69
4.4	Discussion	70
4.5	Moving Forward: AlphaChip’s Broader Impact	70
4.6	External Perspectives from Academic and Industry Leaders	70
4.7	Conclusion	72
	Appendices	73
4.A	Other Discussions	73
4.A.1	Inappropriate Comparison With Commercial Autoplacers	73
4.A.2	Contrived “Ablation” of Initial Placement in Standard Cell Cluster Rebalancing	73
4.A.3	Flawed Study of Correlation Between Proxy Cost and Final Metrics	74
4.A.4	Incorrect Claim of Validation by Google Engineers	75
II	Reinforcement Learning for Language Modeling	76
5	Synthetic Data Generation and Multi-Step RL	77
5.1	Introduction	77
5.2	Methodology	79
5.2.1	Multi-Step Data Collection	80
5.2.2	Step-Wise Reinforcement Learning Methodology	81
5.2.3	Step-Wise Inference-time Evaluation	82
5.3	Related Work	83
5.4	Experiments	85
5.4.1	Evaluation Datasets	85

5.4.2	Results and Discussion	86
5.5	Conclusion	90
Appendices		91
5.A	Prompts for Synthetic Data Generation, Filtering, and Evaluation	91
5.B	Impact of Model Size on Effectiveness of SWiRL	94
5.C	Error Analysis of Three LLM Judges	95
5.D	Example Trajectories	96
5.E	Example Ids	98
6	Benchmarking Multi-Step Reasoning and Tool Use	101
6.1	Introduction	101
6.2	Data Collection	103
6.2.1	Sourcing and Generating Queries	103
6.2.2	Filtering and Rewriting Queries	103
6.2.3	Generating Trajectories with Human Annotators	103
6.3	Evaluation Benchmark	105
6.3.1	LLM as a Judge	105
6.3.2	Data Format	106
6.3.3	Dataset Composition	106
6.4	Dataset Validation and Analysis	106
6.4.1	Dataset Validation	107
6.4.2	Multi-Hop Nature	107
6.4.3	Distribution Over Actions	108
6.5	Rubric-Based LLM Evaluation	108
6.6	Experiments	109
6.7	Related Work	109
6.8	Conclusion	111
Appendices		112
6.A	Filter and Rewrite Input Queries for Multi-Hop Search	112
6.B	Summary	112
6.C	Instructions	112
6.D	Example 1: Accept Query	112
6.E	Example 2: Rewrite Query	113
6.F	Example 3: Query Rejected	113
6.G	Data Collection for Multi-Hop Search & Response Generation	114

6.H Example Trajectories	115
7 Conclusions	120

List of Tables

3.1	Hyperparameters used to finetune the RL agent	42
3.2	Hyperparameters used for force-directed standard cell clustering	42
3.3	Hyperparameters used to generate standard cell clusters with hMETIS	43
3.4	Comparison against baselines	44
3.5	Effect of cost trade-offs on the post-PlaceOpt performance	48
3.6	Sensitivity of results to the choice of random seed	48
3.C.1	Ablation Study on the Effect of Macro Ordering	55
3.D.1	Detailed performance metrics for a TPU-v5 block	57
3.E.1	Performance of our method compared to Simulated Annealing	58
4.2.1	Summary of Cheng et al. Tensorboards	68
4.A.1	IRL results after clustering standard cells with and without the initial placement	74
5.4.1	Comparison of SWiRL accuracy across multiple datasets	85
5.4.2	SWiRL generalization performance	87
5.4.3	Impact of SWiRL on Process Correctness	90
5.C.1	Error Rates for Gemma-2-27b Judgments on HotPotQA (N=100)	95
5.C.2	Manual Analysis of LLM Math Grading Accuracy (N=100)	95
6.3.1	Number of examples in each split of the dataset.	106
6.6.1	Percentage of ‘Excellent’ Grades out of 300 example answers graded on a 5-point Likert scale, where ‘Excellent’ corresponds to the highest possible score.	110
6.6.2	Full Distribution of LLM-Generated Grades	110

List of Figures

2.1	Diagram of RL agent exploring an environment and taking actions	16
2.2	Placement optimization overview	18
3.1	Zero-shot performance of Edge-GNN vs. GCN	30
3.2	AlphaChip Policy and value network architecture	31
3.3	Overview of AlphaChip method and training regimen	32
3.4	Evaluation workflow for producing the results in Table 3.4	41
3.5	Training from scratch vs. fine-tuning for varying amounts of time	45
3.6	Convergence plots on Ariane RISC-V CPU	46
3.7	Effect of pre-training dataset size	47
3.8	Generalization performance as a function of pre-training dataset size	49
3.9	Visualization of Ariane RISC-V CPU placements	50
3.10	Visualization of a TPU placement	50
3.A.1	DQN Variants vs. Human Baseline on an Edge Block	52
3.A.2	PPPO vs. Human Baseline on an Edge Block	53
3.B.1	Placement cost as a function of training times of up to 60 hours	54
3.D.1	Image of an AlphaChip layout taped out in TPU-v5	56
3.E.1	Sample Efficiency of RL vs. SA	59
4.1.1	AlphaChip deployments across three generations of TPU	61
4.2.1	Performance gains from pre-training on a larger number of blocks	65
4.2.2	Pre-training improves convergence speed	66
4.2.3	Figures demonstrating that speed and quality improve with additional compute resources	66
4.2.4	Convergence plots from Cheng <i>et al.</i> 's project site	67
4.A.1	Cheng et al.'s Table 2 shows weak but positive correlation	74
5.2.1	SWiRL Stage 1: Generating and Filtering Multi-Step Synthetic Trajectories	80
5.2.2	SWiRL Stage 2: Step-Wise Reinforcement Learning	81
5.2.3	SWiRL Stage 3: Multi-Step Inference	83

5.4.1 Impact of Data Filtering on Model Performance	86
5.4.2 Comparison of SFT and SWiRL	88
5.4.3 Performance of SWiRL with and without multi-step tool use	89
5.4.4 Performance as a Function of Synthetic Dataset Size	89
5.B.1SWiRL Performance vs. Model Size	94
6.4.1 Histogram depicting the number of hops	108
6.4.2 Piechart depicting the composition of trajectories with respect to action types.	109

I see only one move ahead, but always the best move.

— *Charles Jaffe, Chess Review, May 1946*

Chapter 1

Introduction

1.1 Reinforcement Learning for Sequential Decision-Making

When I started my PhD in 2019, the research community harbored doubts about the reliability and effectiveness of reinforcement learning (RL) methods, questioning whether such methods could be used to solve real-world problems or whether they even worked at all [91]. This gap stemmed largely from the instability of these methods [154, 127], their relative sample inefficiency [11, 193], and the difficulty of attributing reward to individual steps within the long trajectories characteristic of real-world tasks [146, 145, 194, 166, 32].

To address these challenges, my dissertation presents approaches to problem formulation, representation learning, effective reward attribution, and scalable curation of high-quality multi-step trajectories. To illustrate these principles, I will introduce RL agents capable of solving real-world challenges in two disparate areas: chip design and language modeling.

In the first half of this dissertation, I will motivate the use of reinforcement learning for placement optimization, a combinatorial optimization task that is commonly found in systems and chip design. I will then introduce AlphaChip, a deep reinforcement learning method capable of generating superhuman chip layouts in hours, rather than weeks or months of human effort. AlphaChip was one of the first reinforcement learning methods deployed to solve a real-world engineering problem, and it has designed chip layouts in the last four generations of Google TPU, as well as other chips by Alphabet and external chipmakers. Finally, I will discuss the subsequent impact of this method and the research discussions surrounding it.

In the second half, I will introduce SWiRL (Step-Wise Reinforcement Learning), an approach to synthetic data generation and reinforcement learning that improves the ability of large language models (LLMs) to perform multi-step reasoning and tool use. In particular, I target multi-hop question-answering and mathematical reasoning. I will then describe COMPASS-QA, a new dataset

designed for evaluating LLM performance on challenging multi-step reasoning tasks and for training LLM-based RL agents in these domains. I will conclude this dissertation by presenting a new dataset for training and evaluating the performance of LLM-based RL agents on challenging multi-step reasoning tasks, and discuss open problems and opportunities in this new frontier.

Placing the nodes of a chip onto a 2D canvas may seem wildly different from solving mathematical reasoning problems or answering multi-hop questions using a search engine, and yet these tasks share common structure. In all cases, they involve performing a sequence of non-trivial decisions, culminating in a final solution. This dissertation demonstrates that these types of tasks can be effectively addressed by deep neural networks optimized with reinforcement learning, using the approaches developed in this dissertation to mitigate the RL pitfalls outlined above.

My long-term research objective was to build computer systems capable of understanding and generating human language, and as we asymptotically approach this goal, it shifts to designing highly intelligent LLM agents. Compute is the fuel for progress in Natural Language Processing (NLP), as demonstrated particularly clearly by neural scaling laws [83, 106, 85]. I chose to pursue this dissertation topic because I believe that RL for sequential decision-making is a powerful framework that unifies much of my work. Furthermore, my work initiates a recursive feedback loop between hardware and AI, where advances in AI lead to stronger hardware, which leads to stronger AI, and so on. Indeed, TPUs with superhuman chip layouts designed by AlphaChip were used to train and evaluate all of the models and datasets described in the second half of this dissertation.

1.2 A Brief History of Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning, in which an *agent* learns by interacting with an *environment* and receives positive or negative *reward* as a result of its *actions*¹ [193]. In this section, I will provide an abbreviated history of reinforcement learning to contextualize the work presented in this dissertation.

The theoretical underpinnings of reinforcement learning trace back to the early 1900s and are rooted in an unlikely marriage between animal psychology and control theory. In psychology, Edward Thorndike’s “Law of Effect” in the early 20th century proposed that behaviors followed by positive feedback are more likely to be repeated, establishing a fundamental principle of learning from feedback [201]. In control theory and operations research, Richard Bellman’s work on dynamic programming in the 1950s provided a mathematical framework for making optimal sequences of decisions over time [21]. Together, these separate lines of research formed the conceptual foundation of RL, namely learning through trial-and-error interaction with an environment to achieve a goal.

In 1959, Arthur Samuel introduced a checkers-playing program that arguably represents the first

¹See Section 2.3 for a brief introduction to reinforcement learning.

reinforcement learning approach, or at least a clear precursor. This pioneering work shared many key elements of what later came to characterize the field of reinforcement learning. For example, it featured a lookup table containing all board positions ever seen, a scoring function to evaluate how favorable a particular position was, and the ability to adjust its own weights and learn from experience via self-play [175]. However, Samuel’s mathematical formulation differed substantially from subsequent work, taking a more heuristic and practical approach that was distinct from the formalisms of dynamic programming [193].

In the 1980s and early 1990s, many seminal contributions were made to the field of reinforcement learning, such as the introduction of the actor-critic architecture [20], temporal-difference (TD) learning [194], and other model-free algorithms such as Q-Learning [217] and SARSA [174], which enabled learning without an explicit model of the environment. Although these works introduced key concepts that are still relevant today, they relied on exhaustive lookup tables covering all possible states. As a result, for problems with large or continuous state spaces, initializing or maintaining such a table is computationally infeasible [13, 193].

In 1995, TD-Gammon took a first step toward overcoming this fundamental limitation. In particular, this work combined TD-learning with a neural network approximating the value function for the game of backgammon [200]. TD-Gammon kicked off the modern era of deep reinforcement learning, where deep neural network representations replaced exhaustive lookup tables. However, the combination of function approximation, bootstrapping (learning from estimates), and off-policy training proved to be a notoriously unstable recipe for divergence—‘the Deadly Triad’ as it is now known [193]—and triggered an RL winter in which other machine learning approaches, such as supervised learning, enjoyed much faster progress and wider adoption [205, 152].

In a watershed moment for the field, AlphaGo defeated the human world champion at the game of Go in 2016 [186], revitalizing interest in deep reinforcement learning methods. Unfortunately, despite promising subsequent results in games and simulations [186, 187, 209, 105, 89], RL methods had yet to be successfully deployed to solve real-world problems. For example, even in the case of the well-known RL for datacenter cooling paper which suggested that power consumption could be reduced by up to 40% [172], the RL method itself was never deployed in production and the performance gains reported were merely hypothetical.

In 2020, my collaborators and I introduced AlphaChip, a deep reinforcement learning approach to designing chip layouts [16, 149]. AlphaChip was arguably the first RL method deployed to solve a real-world engineering problem, and its superhuman chip layouts have been taped out in the last four generations of Google’s Tensor Processing Units (TPUs), as well as other chips across Alphabet and by external chipmakers. These chips have been deployed and are currently running in Google datacenters all over the world [74]. This work will be discussed in detail in Part I of my dissertation.

AlphaChip’s publication triggered an explosion of work on AI for chip design [229, 227, 233,

228, 42, 66, 23, 195, 29, 47, 65, 216, 28, 144]. Nevertheless, as described in Sutton’s “The Bitter Lesson” [192], there is often reluctance to accept the application of machine learning to new areas, and ultimately this led to some confusion around our work, which I will discuss in Chapter 4. Today, RL approaches to chip design have been widely adopted by leading commercial Electronic Design Automation (EDA) companies, such as Synopsys and Cadence, and heavily featured at top chip design conferences like the Design Automation Conference (DAC)².

In recent years, the natural language processing (NLP) community has achieved impressive results via reinforcement learning optimization of large language models (LLMs) [45, 18, 199, 69, 12, 161]. In Part II of my dissertation, I will describe some of my work on this new frontier, where I have focused on building agents that can reason, plan, and use tools over multiple steps to accomplish increasingly complex tasks. Ultimately, I believe that the full potential of RL lies in multi-step sequential decision-making, and the development of agentic AI systems capable of autonomously solving a wide variety of real-world challenges.

1.3 Thesis Statement, Research Questions, and Contributions

Overall, my thesis is that reinforcement learning is a powerful technique whose weaknesses can be overcome through careful problem formulation (design of the action space), representation learning (design of the neural state representation), curation of high-quality synthetic data (to enable generalization across tasks), and per-step reward attribution (for more stable and effective learning). When wielded properly, RL methods are capable of having real-world impact across a broad range of disciplines, including chip layout optimization and multi-step reasoning in large language models.

To this end, my dissertation seeks to provide insights into the following research questions:

- *How can we develop RL strategies that are effective across long time horizons and multiple steps of sequential decision-making?*
- *What are effective strategies to mitigate instability in RL training and enable real-world impact?*
- *What approaches can be employed to address the relative sample inefficiency of RL methods?*

In the chapters that follow, I will discuss in detail the key lessons learned in the development and successful deployment of RL agents across a broad range of domains and modalities, including both chip design and language modeling. Below I summarize these findings:

Handling Long Trajectories: In the course of my dissertation work, I found that effective reward attribution was key to scaling RL to the long trajectories characteristic of real-world tasks. For

²At DAC 2025, there is even a session entitled “Need a Break from AI? Memory-centric Computing for Beyond Machine Learning Application”, jokingly alluding to the fact that nearly every session is now AI-related.

example, in SWiRL, we decomposed synthetic trajectories into single actions (chains of thought followed by tool use invocations) in the context of all prior actions, enabling the RL policy to receive a rich and granular reward signal for each action. In contrast, for AlphaChip, we found that it was not necessary to provide granular reward signal after each step (though we did explore this approach), but instead focused on decomposing the task into placing a sequence of approximately 100 macros (the most difficult part, which was previously performed by human experts), followed by invocation of a standard cell placement tool to quickly and approximately place millions of standard cells (a problem that was already effectively addressed by prior analytic approaches). This allowed us to reduce the effective sequence length and enabled the effective deployment of an RL approach to chip placement optimization with only implicit per-step signal via AlphaChip’s value network and the edge-based graph neural network encodings of the state at each step.

Mitigating RL Instability: I found that both problem formulation (i.e., the design of the state and action space) and representation learning (via either custom architecture and/or large-scale pre-training) were critical to addressing the instability of RL training and enabling effective deployment. Regarding pre-training and representation learning, we observed that this created a dramatic difference in training stability and convergence of AlphaChip. For example, AlphaChip policies trained from scratch had approximately a 1 in 10 chance of converging, whereas those initialized from a pre-trained policy nearly always converged. This underscores the value of starting with a well-initialized representation. My experience with SWiRL echoed this finding: initializing from a pre-trained Large Language Model (LLM) provided a strong foundation, leading to remarkably stable RL training.

Beyond representation learning, problem formulation emerged as a critical factor for enhancing RL stability. SWiRL achieved more consistent convergence by gathering trajectories offline and subsequently employing LLMs to filter this data, disentangling the unstable tool use invocation from the RL training. AlphaChip implemented an even more direct offline approach: chip layouts were generated and evaluated, with only high-quality layouts being retained. This inherently improved stability because the quality of the best-known layout could only monotonically increase throughout training, providing a more consistent signal, albeit a noisy one due to imperfect proxy costs.

Addressing Sample Inefficiency: Finally, handling RL’s infamously poor sample inefficiency required a combination of designing custom neural architectures with the appropriate inductive bias and developing new approaches to scaling synthetic data. In AlphaChip, we developed a novel edge-based graph neural network encoder that enabled us to effectively represent a range of input chips and to generalize across them. This dramatically improved the sample efficiency of our RL approach to chip floorplanning, causing it to be approximately 100× more sample efficient than prior approaches like simulated annealing. SWiRL offered a complementary lesson: if a model

struggles with sample inefficiency, one approach is to simply generate more samples. We found that scaling the generation of synthetic trajectories consistently improved performance at all tested scales. Importantly, these improvements generalized across different tasks and tools, highlighting the power of abundant, targeted synthetic data for overcoming sample limitations in LLM-based agents.

1.4 Overview of Dissertation

This dissertation is divided into two main parts. In Part I, I will describe reinforcement learning approaches to problems in systems and chip design. In Part II, I will explore how reinforcement learning methods can be developed to overcome challenges in natural language processing. Here, I provide an overview of each chapter.

Part I: Reinforcement Learning for Chip Design

The first part of this thesis focuses on RL agents for placement optimization and chip design, framing these tasks as sequential decision-making problems that can be effectively addressed using reinforcement learning.

Chapter 2: Placement Optimization with Reinforcement Learning. This chapter begins with an overview of reinforcement learning, followed by a description of placement optimization, a challenging category of combinatorial optimization that is a common subtask in systems and chip design tasks. I will then motivate the use of reinforcement learning in this domain and describe how it can be best adapted to this category of problem, including considerations for the design of the action space, neural representation of state, reward function, and the handling of hard constraints.

Chapter 3: AlphaChip: A Graph Placement Methodology for Fast Chip Design. In this chapter, we will describe AlphaChip, a deep reinforcement learning method capable of generating superhuman chip layouts in hours, rather than weeks or months of effort by teams of human experts. AlphaChip formulates chip placement as a sequential reinforcement learning problem, placing the nodes of a chip one-at-a-time onto a 2D canvas, representing the state using a novel edge-based graph neural network, and calculating reward signal as the weighted average of approximate wirelength, density, and routing congestion.

Chapter 4: From Scrutiny to Silicon: Post-Publication Discussion and Impact of AlphaChip. In this chapter, we will describe the real-world impact of AlphaChip, including its deployment in four generations of TPU, datacenter CPUs (Axion), and other chips across Alphabet and by external chipmakers. As one of the first RL methods deployed to solve a real-world engineering

problem, its publication triggered an explosion of work on AI for chip design, as well as intense skepticism and discussion within the semiconductor community, which will be summarized in this chapter.

Part II: Reinforcement Learning for Language Modeling

The second part of this thesis develops RL agents and datasets for multi-step reasoning and tool use in the domain of natural language processing, focusing on question answering and mathematical reasoning. I also introduce a new dataset designed to enable evaluation and training of future LLM-based RL agents in this domain.

Chapter 5: Synthetic Data Generation and Multi-Step Reinforcement Learning This chapter introduces Step-Wise Reinforcement Learning (SWiRL), our approach to addressing limitations of traditional single-step reinforcement learning (RL) methods for complex, multi-step language model tasks. SWiRL employs iterative synthetic data generation and a step-wise decomposition of trajectories for optimization. We demonstrate its effectiveness with substantial performance gains (e.g., up to 21.5% relative accuracy improvement) on diverse benchmarks and highlight its capacity for cross-task generalization.

Chapter 6: Benchmarking Multi-Step Reasoning and Tool Use In this chapter, we introduce a new benchmark to measure the ability of current models to perform multi-step reasoning and tool use. Furthermore, we describe a rubric-based LLM evaluation framework which enables consistent evaluation of open-ended questions. We provide 700 challenging questions that require multiple steps of reasoning and tool use to effectively answer. To enable training RL agents with these capabilities, we provide trajectories generated by human annotators outlining the steps they took to reach the final answer. We hope that this dataset will help researchers meaningfully benchmark the effectiveness of new models on multi-step reasoning and tool use.

1.5 Bibliographic Notes

The chapters of my dissertation are largely based on the following publications, where * indicates co-first authorship. For clarity, I describe my personal contributions to each co-authored publication.

- **Chapter 2**

- **Publication:** Anna Goldie & Azalia Mirhoseini, Placement Optimization with Deep Reinforcement Learning. Association for Computing Machinery, Proceedings of the International Conference on Physical Design 2020.

- **Contributions:** I conceived the project with Azalia Mirhoseini, drew insights from experiments and analysis that I had run across various deep RL for placement optimization problems (e.g. device placement and chip floorplanning), and led the writing of the paper.

- **Chapter 3**

- **Publication:** Azalia Mirhoseini*, Anna Goldie*, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean. A Graph Placement Methodology for Fast Chip Design. *Nature* 594, 207–212 (2021).
- **Contributions:** I conceived the idea and initiated the project with Azalia Mirhoseini and Jeff Dean, ran experiments, pair-programmed the first RL prototype with Azalia, generated all of the superhuman layouts used in AlphaChip’s first TPU deployment, supervised other researchers and engineers, proposed ideas, analyzed results, designed experiments, and wrote the *Nature* paper with Azalia, as well as the cover letter and all of the peer review responses throughout the 7-month review process [73]. Note that the two co-first authors contributed equally, and the order of their names was determined by a coin flip [75].

- **Chapter 4:**

- **Publication:** Anna Goldie*, Azalia Mirhoseini*, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean. Addendum: A Graph Placement Methodology for Fast Chip Design. *Nature* 634, E10-11 (2024).
- **Contributions:** I devised the strategy for the Addendum, including how we would describe the scientific disagreement, how we would address all concerns raised by *Nature* and the peer reviewers, and how we would highlight the subsequent impact of this work. I proposed the experiments in the Addendum and supervised their execution. I wrote the Addendum with input from Azalia and Jeff and sign-off from all 20 of the original *Nature* co-authors.
- **Publication:** Anna Goldie, Azalia Mirhoseini, & Jeff Dean. That Chip Has Sailed: A Critique of Unfounded Skepticism Around AI for Chip Design. arXiv 2024.
- **Contributions:** I originated the idea (and the name) of this paper, obtained approval to publish it, and wrote the paper with input from Jeff Dean and Azalia Mirhoseini.

- **Publication:** Anna Goldie, Azalia Mirhoseini. How AlphaChip Transformed Chip Design. Google DeepMind Blog. September 26, 2024.
- **Contributions:** I proposed the idea and the structure of the blogpost, wrote the initial draft, and negotiated with DeepMind comms on every sentence to maintain scientific accuracy. I also coordinated with external research leaders and internal partners.

- **Chapter 5:**

- **Publication:** Anna Goldie*, Azalia Mirhoseini*, Hao Zhou, Irene Cai, & Christopher D. Manning. Synthetic Data Generation and Multi-Step RL for Reasoning and Tool Use. Under Review at the Conference on Language Modeling (COLM) 2025.
- **Contributions:** I conceived the idea for this project in collaboration with Azalia and Chris Manning. I ran all of the experiments and wrote all of the code. I wrote the paper with Azalia, and prepared the peer review responses.

- **Chapter 6:**

- **Publication:** Anna Goldie*, Azalia Mirhoseini*, Hao Zhou, Christopher D. Manning. Benchmarking Multi-Step Reasoning and Tool Use in Large Language Models. Under Embargo until October 18, 2025.
- **Contributions:** I initiated the project and conceived the idea in collaboration with Azalia and Chris Manning. I oversaw data collection, led trajectory filtering, the development and implementation of the design, and did all the experiments and writing.

During my PhD, I also contributed to the following publications, which are not included in my dissertation.

- Anna Goldie, Azalia Mirhoseini. RL for Placement and Partitioning, Textbook Chapter in RL for Placement and Partitioning Machine Learning Applications in Electronic Design Automation, Springer International Publishing. Aug 10, 2022.
- Caia Costello, Simon Guo, Anna Goldie, Azalia Mirhoseini. Think, Prune, Train: Can Small Models Teach Themselves to Reason? ICLAD 2025.
- Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, Christopher D. Manning. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. ICLR 2024
- Moussa Doumbouya, Ananjan Nandi, Gabriel Poesia, Davide Ghilardi, Anna Goldie, Federico Bianchi, Dan Jurafsky, Christopher D. Manning. h4rm3l: A language for Composable Jailbreak Attack Synthesis. International Conference on Learning Representations (ICLR) 2025.

- Parth Sarthi, Shreyansh Choudhary, Ayush Mangal, Ankur Bhatia, Anna Goldie, Suvrat Bhooshan, Christopher D Manning. Scaling Compute-Optimal Text-to-Speech Models. Under Review at the International Conference on Machine Learning (ICML) 2025.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon *et al.* Constitutional AI: Harmlessness from AI Feedback. arXiv 2022.
- Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, Azalia Mirhoseini. A Full-Stack Search Technique for Domain Optimized Deep Learning Accelerators. Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, p27-42. February 28, 2022.
- Summer Yue, Ebrahim M Songhori, Joe Wenjie Jiang, Toby Boyd, Anna Goldie, Azalia Mirhoseini, Sergio Guadarrama. Scalability and Generalization of Circuit Training for Chip Floorplanning. Proceedings of the 2022 International Symposium on Physical Design, p65-70. April 13, 2022.
- Ahmet F Budak, Zixuan Jiang, Keren Zhu, Azalia Mirhoseini, Anna Goldie, David Z Pan. Reinforcement Learning for Electronic Design Automation: Case Studies and Perspectives. 2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC), p500-505. January 17, 2022.
- Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, David Z Pan. Delving into Macro Placement with Reinforcement Learning. 2021 ACM/IEEE 3rd Workshop on Machine Learning for Computer-Aided Design (MLCAD), p1-3.
- Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, Azalia Mirhoseini. Clustering by Learning to Optimize Normalized Cuts. NeurIPS 2019 Workshop on Sets & Partitions.
- Deep Ganguli, Amanda Askell, Nicholas Schiefer, Thomas I. Liao, Kamilė Lukošiuūtė, Anna Chen, Anna Goldie, Azalia Mirhoseini, Catherine Olsson, Danny Hernandez *et al.* The Capacity for Moral Self-Correction in Large Language Models. arXiv 2023.
- Sandipan Kundu, Yuntao Bai, Saurav Kadavath, Amanda Askell, Andrew Callahan, Anna Chen, Anna Goldie, Avital Balwit, Azalia Mirhoseini, Brayden McLean *et al.* Specific versus General Principles for Constitutional AI. arXiv 2023.
- Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiuūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini *et al.* Measuring Progress on Scalable Oversight for Large Language Models. arXiv 2022.

- Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, James Laudon. A Generalized Approach to End-to-End Computational Graph Optimization. NeurIPS 2020 (Oral)
- Jialin Song, Joe Jiang, Ebrahim Songhori, Anna Goldie, Navdeep Jaitly, Azalia Mirhoseini. Policy Optimization By Local Improvement Through Search. ICML 2020 Workshop on Inductive Biases, Invariances and Generalization in RL.

Part I

Reinforcement Learning for Chip Design

Chapter 2

Placement Optimization With Reinforcement Learning

2.1 Background

Placement Optimization is an important problem in systems and chip design, which consists of mapping the nodes of a graph onto a limited set of resources to optimize for an objective, subject to constraints. Common examples of this class of problem include placement of TensorFlow graphs onto hardware devices to minimize training or inference time, or placement of an ASIC or FPGA netlist onto a grid to optimize for power, performance, and area. In this chapter, we start by motivating reinforcement learning (RL) as a solution to the placement problem. We then give an overview of what deep reinforcement learning is. We next formulate the placement problem as a reinforcement learning problem, and show how this problem can be solved with policy gradient optimization. Finally, we describe lessons we have learned from training deep reinforcement learning policies across a variety of placement optimization problems.

2.2 Placement Optimization as an RL Problem

Placement optimization is a very challenging problem as several factors, including the size and topology of the input graph, number and properties of available resources, and the requirements and constraints of feasible placements all contribute to its complexity. Placement optimizations of this form appear in a wide range of science and engineering applications, including hardware design [139], city planning [14], vaccine testing and distribution [142], and cerebral cortex layout [43]. There are many approaches to the placement problem. A range of algorithms including analytical approaches [113, 136, 48, 131], genetic and hill-climbing methods [50, 58, 117], Integer Linear Programming

(ILP) [225, 33], and problem-specific heuristics have been proposed [184].

More recently, a new type of approach to the placement problem based on deep Reinforcement Learning (RL) [150, 147, 239] has emerged. RL-based methods bring new challenges, such as interpretability, brittleness of training to convergence, and unsafe exploration [55, 91]. However, they also offer new opportunities, such as the ability to leverage distributed computing, ease of problem formulation, end-to-end optimization, and domain adaptation, meaning that these methods can potentially transfer what they learn from previous problems to new unseen instances.

2.3 Principles of Deep Reinforcement Learning

Most successful applications of machine learning are examples of supervised learning, where a model is trained to approximate a particular function, given many input-output examples (e.g., given many images labeled as cat or dog, learn to predict whether a given image is that of a cat or a dog). Today’s state-of-the-art supervised models are typically deep learning models, meaning that the function approximation is achieved by updating the weights of a multi-layered (deep) neural network via gradient descent against a differentiable loss function.

Reinforcement learning, on the other hand, is a separate branch of machine learning in which a model, or policy in RL parlance, learns to take actions in an environment (either the real-world or a simulation) to maximize a given reward function. One well-known example of reinforcement learning is AlphaGo [186], in which a policy learned to take actions (moves in the game of Go) to maximize its reward function (number of winning games). Deep reinforcement learning is simply reinforcement learning in which the policy is a deep neural network.

RL problems can be reformulated as Markov Decision Processes (MDPs) [193]. MDPs rely on the Markov assumption, meaning that the next state s_{t+1} depends only on the current state s_t , and is conditionally independent of the past.

$$P(s_{t+1}|s_0...s_t) = P(s_{t+1}|s_t)$$

Like MDPs, RL problems are defined by five key components:

- states: the set of possible states of the world (e.g., the set of valid board positions in Go)
- actions: the set of actions that can be taken by the agent (e.g., all valid moves in a game of Go)
- state transition probabilities: the probability of transitioning between any two given states.
- reward: the objective to be maximized, subject to future discounting as defined below (e.g., 1 if you win the game of Go, 0 otherwise)

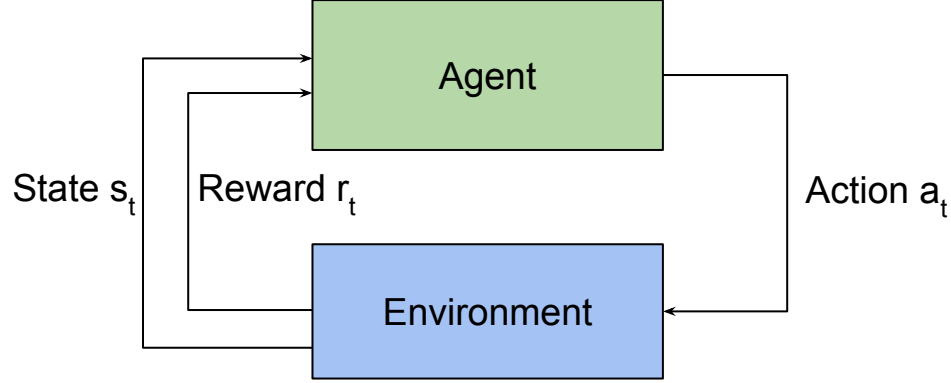


Figure 2.1: In Reinforcement Learning, an agent explores the environment and finds actions to maximize cumulative rewards.

- discount for future rewards: how much to discount the value of future reward, due to its relative uncertainty (e.g., a discount factor of .95 would mean that 95 dollars today is equivalent to 100 tomorrow).

At each time step t , the agent begins in state (s_t) , takes an action (a_t) , arrives at a new state (s_{t+1}) , and receives a reward (r_t) from the environment, as shown in Figure 2.1. Through repeated episodes (sequences of states, actions, and rewards), the agent learns to take actions that will maximize cumulative reward.

Reinforcement learning approaches can be divided into two broad categories: model-free and model-based. In model-free reinforcement learning, we train a policy to take actions that maximize reward from a black-box environment. In model-based reinforcement learning, we train a policy to take actions that maximize reward, while also training an explicit model of the world, which learns to predict the reward and state transitions of the environment. Most existing work on reinforcement learning for systems problems has taken a model-free approach, as it is generally easier to train to convergence. However, model-based reinforcement learning has been shown to be more sample efficient in other domains [92], so it may be a viable direction to take in future work, especially in situations where the reward function is very expensive to evaluate.

Since the agent’s goal is to maximize cumulative reward, one approach is to learn a value function that can predict the reward given a state, $v(s)$, and then take the action which will bring the agent into a state that yields the highest expected reward. However, a more common approach in recent years is to use policy gradient methods, which seek to directly learn the policy $\pi(a|s)$ that predicts the optimal action given the current state. Popular policy gradient methods include REINFORCE [221], A3C [153], TRPO [178], and PPO [179].

RL is helpful in cases where we do not have sufficient labeled data (input-output examples) to take a supervised learning approach or when the objective function is not differentiable. It is also

well-suited to massive search problems, where exhaustive or heuristic-based methods cannot scale, and cases where it is necessary to perform long rollouts in order to determine final reward, such as AlphaGo [186], AlphaStar [209], and OpenAI Five DOTA [160].

Reinforcement learning policies are famously difficult to train, as they tend to be brittle with respect to their hyperparameters, hard to interpret and debug, and prone to catastrophic failures and unsafe exploration. This area of machine learning is not yet as well understood, with few educational resources available, as compared to more established areas like deep learning. This is part of our motivation for writing this chapter, to demystify this area and empower more systems experts to move into this promising, but challenging area.

2.4 Deep Reinforcement Learning for Placement Optimization

In this section, we first start by formulating the placement problem. We will then show how RL can be applied to solve this problem.

2.4.1 Placement Problem Formulation

Recall that the placement problem consists of mapping the nodes of an input graph onto a limited set of resources, such that an objective function is optimized while adhering to any hard constraints. Let us assume the input graph g has nodes v_1, v_2, \dots, v_N . We want to place these nodes onto placement locations $l = l_1, l_2, \dots, l_M$. In this set up, we use RL to find a mapping

$$(v_1, v_2, \dots, v_N) \rightarrow (l_1, l_2, \dots, l_M)$$

that maximizes a reward function R subject to constraints. Here, there is a unique placement location for each node v_i , but each location l_j can be assigned to multiple nodes. The constraints vary by problem, but a common constraint is limited capacity for each placement location, meaning that there is a limit on how many nodes can be assigned to each location. In the next section, we will discuss some case-specific constraints and ways to incorporate them.

To formulate placement as a policy optimization problem, [150] proposed relaxing the objective function. Instead of finding the absolute best placement, one can train a policy that generates a probability distribution of nodes to placement locations such that it maximizes *the expected reward* generated by those placements.

Let us denote the policy π parameterized by θ as π_θ . θ represents the weights of a deep network architecture. Here, we describe the objective, which is to train parameters θ such that the network predicts placement decisions for the nodes of the input graph g , and as a result, the placement reward $R_{l,g}$ is maximized. We can write the cost function we are optimizing for as follows:

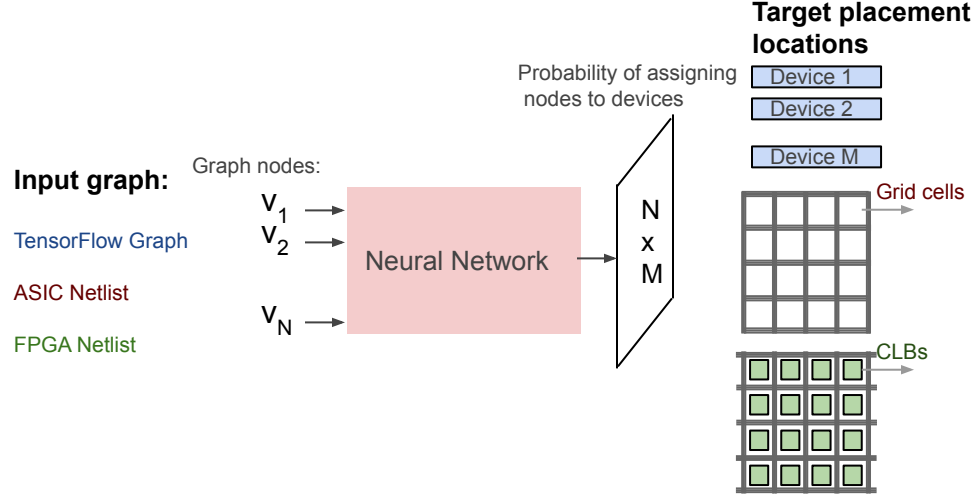


Figure 2.2: Placement optimization overview. Target placement locations for TensorFlow graphs, ASIC netlists, and FPGA netlists are the computing devices (e.g., TPU or GPUs), grid cells of the chip canvas, and FPGA Configurable Logic Blocks (CLBs), respectively.

$$J(\theta, g) = E_{g, l \sim \pi_\theta} [R_{l,g}] = \sum_{l \sim \pi_\theta} \pi_\theta(l|g) R_{l,g} \quad (2.1)$$

We can then train this policy (optimize parameters θ) using a policy gradient based method, which we will discuss in this section.

Here, we provide an overview of the state and action space, reward function and the neural network architecture of the policy by delving into example placement problems, namely TensorFlow device placement, ASIC netlist placement, and FPGA netlist placement. As shown in Figure 2.2, all of these placement problems require mapping the nodes of a graph onto placement locations such that their corresponding reward metrics are optimized. Target placement locations for TensorFlow graphs, ASIC netlists, and FPGA netlists are the computing devices (e.g., TPU or GPUs), grid cells of the chip canvas, and FPGA Configurable Logic Blocks (CLBs), respectively.

For each of these problems, the neural network policy receives a state as input, and outputs an action for that state. In general, the state should represent the information that the policy needs for prediction, such as node ID, type, and adjacency matrix. The network then outputs a probability distribution representing the probability of assigning an input node onto each placement location. The action is selected by sampling or taking the argmax of the output probability distribution.

The reward function varies for different problems. For example, for TensorFlow graph placement, we use negative runtime of a training step of the placed deep network model. For ASIC and FPGA netlists, the reward is more complex and should include various metrics related to power and timing (e.g., total wirelength, routability congestion, and cell density).

2.4.2 Graph Convolutional Neural Networks

As discussed, many placement problems take input in the form of graph. The way in which these input graphs are represented has great impact on the ability of machine learning models to generate high-quality placements. More meaningful representations also help models to learn patterns that generalize to new unseen graphs, as opposed to merely memorizing the graphs that they encounter. We therefore describe some of the recent advances in neural graph representations, such as Graph Convolutional Networks.

Graph neural networks can be divided into four high-level categories [223]: recurrent graph neural networks (RecGNNs) [76, 177, 67], convolutional graph neural networks (ConvGNNs) [26, 82, 54], graph autoencoders (GAEs), and spatial-temporal graph neural networks (STGNNs). RecGNNs preceded ConvGNNs and pioneered the idea of message passing, or representing a node as an iterative aggregate of its neighbors. Each iteration of message passing results in one additional hop (e.g., running the algorithm for k iterations would result in each node being influenced by all neighbors within a k -hop radius). As such, these methods better encode the overall topology of the graph, enabling domain adaptation, as described in the following section. Most graph neural network methods used in systems today are ConvGNNs, which generalize the concept of convolution. After all, images are merely a special case of graphs, in which pixels are nodes connected to the pixels (nodes) immediately surrounding them. ConvGNNs use deep convolutional networks to capture even distant relationships within the graph.

2.4.3 Domain Adaptation

Domain adaptation in placement is the problem of training policies that can learn across multiple graphs and transfer the acquired knowledge to generate more optimized placements for new unseen graphs. In the context of the three examples we discussed in this chapter, domain adaptation means we train a policy across a set of TensorFlow graphs, ASIC or FPGA netlists and apply the trained policy to an unseen TensorFlow graph, ASIC or FPGA netlist. [239] proposed the problem formulation for a domain adaptive policy as follows:

$$J(\theta, G) = \frac{1}{K} \sum_{g \sim G} E_{g, l \sim \pi_\theta} [R_{l, g}] \quad (2.2)$$

In this case, G is a set of K training graphs. The training procedure can be similar to that of traditional machine learning (e.g., using a heldout validation set or cross validation).

2.4.4 Solving the Placement Objective with Policy Gradient Optimization

Now that we have defined an objective function, we will explain how to use *policy gradient optimization* to learn the parameters θ of the policy. We can write the derivative of the objective function in Equation 2.2 as follows:

$$\nabla_{\theta} J(\theta, g) = \sum_{l \sim \pi_{\theta}} \nabla_{\theta} \pi_{\theta}(l|g) R_{l,g} \quad (2.3)$$

$$= \sum_{l \sim \pi_{\theta}} \pi_{\theta} \frac{\nabla_{\theta} \pi_{\theta}(l|g)}{\pi_{\theta}} R_{l,g} \quad (2.4)$$

$$= \sum_{l \sim \pi_{\theta}} \pi_{\theta} \nabla_{\theta} \log(\pi_{\theta}(l|g)) R_{l,g} \quad (2.5)$$

$$= E[\nabla_{\theta} \log(\pi_{\theta}(l|g)) R_{l,g}] \quad (2.6)$$

The equation above is the basis of various policy gradient optimization methods, such as REINFORCE [221], PPO [179], and SAC [81].

2.5 Ingredients for RL Success

In this section, I will share some of the lessons that I have learned in training deep reinforcement learning policies to solve placement problems in computer systems and chip design.

Reward Function: Designing the right reward function is one of the most critical decisions. Some properties of effective reward functions are as follows:

1) Reward functions should be fast to evaluate; RL training often requires 10-100s of thousands of iterations of reward evaluation before reaching convergence. While the exact timing that makes a tractable reward function depends on the complexity of the problem, a sub-second reward function would be effective in nearly any scenario.

2) Reward functions should be strongly correlated with the true objective. In many real-world scenarios, we need to use simulators or proxy reward functions to approximate the true objective, which may be prohibitively expensive to calculate. If the proxy reward is not well-correlated with the true objective, we are solving the wrong problem and the learned placement is unlikely to be useful. While designing a good simulator or approximate function is a challenging task in its own right, it is helpful to build a reward function by combining various approximate metrics that each independently correlate with the true reward. For example, for TensorFlow placement, the proxy reward could be a composite function of total memory per device, number of inter-device (and therefore expensive) edges induced by the placement, imbalance of computation placed on each device. By incorporating a weighted average of multiple proxy rewards, the total variance of the reward error is reduced and

over-fitting to a particular proxy metric is avoided.

3) Another important factor is correctly engineering the reward function. This could be as simple as normalizing the reward or applying more complex functions to change the shape of the reward. For example, for the device placement problem, measuring the runtime of one step of the TensorFlow graph was the true reward function. Due to the runtime varying widely across different placements, using the runtime directly would interfere with learning and gradient updates. We chose to instead use the square root of the runtime, which effectively dampened the range of values.

Action Space: Another key ingredient is designing the appropriate action space. For example, the problem could be formulated as placing the nodes of the netlist one at a time onto the chip netlist, or as placing all of the nodes and then deciding which perturbation (e.g., swap, shift, rotate, etc.) to apply to each of the nodes in a fixed sequence. In device placement, we chose to place all of the TensorFlow nodes onto hardware devices before evaluating the reward for that placement, because otherwise measuring the reward of a partial placement would be very difficult, if not impossible. For ASIC placement, on the other hand, one can define partial reward functions, because it is possible to measure changes in metrics, such as wirelength and congestion, as nodes are being placed.

Managing Constraints: The constraints for feasible placements vary across placement problems. For example, a common constraint is the capacity of placement locations, which limits the number of nodes that can be placed onto that location. For example in device placement, the memory footprint of the nodes placed onto a single device should not exceed the memory limit of that device. Another constraint is that certain nodes cannot co-exist on the same location. For example, in ASIC placement, two macro blocks cannot overlap on the chip canvas.

There are many approaches to enforcing these constraints to avoid or reduce the number of infeasible placements generated by the policy. Perhaps the most straightforward way to handle the constraints is to penalize the policy with a large negative reward whenever it generates infeasible placements. A challenge with this solution is that the policy does not gain any information about how far this placement was from a feasible placement. In the most extreme case, if all of the initial placements generated by the policy are infeasible, there will be no positive signal to teach the policy how to explore the environment and training will fail. Thus, creating a reward function that penalizes the infeasible placements relative to how far they are from viable placements becomes critical.

Another approach is to force the policy to only generate feasible placements. This can be accomplished via a function that masks out the infeasible placements. For example, a mask can be updated given the partial placement of the graph nodes. Each time a new node is placed, the density of all the locations is updated (based on the locations of the nodes that are already placed). The action space then becomes limited to those locations that have enough free capacity to accept the new node. This approach has its own challenges as calculating the mask, similar to calculating the reward, must be done efficiently.

Representations: Finally, the way in which state is represented has significant impact on the performance of the policy and its ability to generalize to unseen instances of the placement problem. For example, in the earlier TensorFlow device placement papers [150, 147], we represented the computational graph as a list of adjacencies, indices of the node operations, and sizes of the operations. This approach was effective when the policy was trained from scratch for each new TensorFlow graph, but was unable to generalize or transfer what it learned to new graphs. On the other hand, [239] used graph convolutional neural networks to learn better representations of the computational graph structure, and was able to transfer knowledge across graphs.

2.6 Conclusion

In this chapter, we discuss placement optimization with deep reinforcement learning. Deep RL is a promising approach for solving combinatorial problems, and enables domain adaptation and direct optimization of non-differentiable objective functions. Training RL policies is a very challenging task, in part due to the brittleness of gradient updates and the costliness of evaluating rewards. In this chapter, we provide an overview of deep RL, formulate the placement problem as an RL problem, and discuss strategies for training successful RL agents.

We predict a trend towards more effective RL-based domain adaptation techniques, in which graph neural networks will play a key role in enabling both higher sample efficiency and more optimal placements. We also foresee a future in which easy-to-use RL-based placement tools will enable non-ML experts to harness and improve upon these powerful techniques.

Chapter 3

AlphaChip: A Graph Placement Methodology for Fast Chip Design

3.1 Background

In Chapter 2, we described placement optimization and motivated an RL approach to this combinatorial optimization task. In this chapter, we describe AlphaChip, an RL methodology for chip placement optimization. Chip floorplanning is the engineering task of designing the physical layout of a computer chip. Despite five decades of research [139], chip floorplanning has defied automation, requiring months of intense effort by physical design engineers to produce manufacturable layouts. In this chapter, we present a deep reinforcement learning (RL) approach to chip floorplanning. In under six hours, our method automatically generates chip floorplans that are superior or comparable to humans in all key metrics, including power consumption, performance, and chip area. To achieve this, we pose chip floorplanning as a reinforcement learning problem, and develop a novel edge-based graph convolutional neural network architecture capable of learning rich and transferable representations of the chip. As a result, our method is capable of leveraging past experience to become both better and faster at solving new instances of the problem, ushering in a new era in which chip design is performed by artificial agents with more experience than any human designer could ever gain. Our method has been used in the design of the last four generations of Google TPU, including every generation since the method was published in *Nature*, saving thousands of hours of human effort and hundreds of millions of dollars in labor, compute, and opportunity cost. Finally, we believe that more powerful AI-designed hardware will fuel advances in AI, creating a symbiotic relationship between the two fields.

3.2 Introduction

In this chapter, we propose a new graph placement method based on reinforcement learning, and demonstrate state-of-the-art results on chip floorplanning, an NP-hard problem [198]. Our method generates manufacturable chip floorplans in under 6 hours, compared to the strongest baseline which requires months of intense effort by human experts.

A computer chip is divided into dozens of blocks, each of which is an individual module, such as a memory subsystem, compute unit, or control logic system. These blocks can be described by a netlist, a graph of circuit components, such as macros (memory components) and standard cells (logic gates like NAND, NOR, and XOR), all of which are connected by wires. Chip floorplanning involves placing netlists onto chip canvases (2D grids), such that performance metrics (e.g., power consumption, timing, area, and wirelength) are optimized, while adhering to hard constraints on density and routing congestion.

Since the 1960s, many approaches to chip floorplanning have been proposed, falling into three broad categories: partitioning-based methods [25, 61, 173], stochastic/hill-climbing approaches [118, 180, 176], and analytic solvers [138, 87, 211, 111, 136, 48]. However, none of these approaches could achieve human-level performance, and the exponential growth in chip complexity has rendered these techniques largely unusable on modern chips.

The limitations of these prior approaches are varied. For example, partitioning-based methods sacrifice quality of the global solution in order to scale to larger netlists, and a poor early partition may result in an unsalvageable final result [214, 102]. Hill-climbing approaches have low convergence rates and do not scale to modern chip netlists with millions or billions of nodes [136]. Prior to this work, analytic solvers were the leading approach, but they can only optimize for differentiable loss functions, meaning that they cannot effectively optimize for critical metrics, such as routing congestion or timing violations. Our method, on the other hand, can scale to netlists with millions of nodes, and optimizes directly for any mixture of differentiable or non-differentiable cost functions. Furthermore, our method improves in both speed and quality of result as it is exposed to more instances of the chip placement problem.

Due to the limitations of these prior methods, human physical designers must iterate for months with commercial EDA tools, taking as input a register transfer level (RTL) description of the chip netlist, generating a manual placement of that netlist onto the chip canvas, and waiting up to 72 hours for EDA tools to evaluate that placement. Based on this feedback, the human designer either concludes that the design criteria have been achieved, generates an updated floorplan for evaluation, or provides feedback to upstream RTL designers who then modify the low-level code to make the placement task easier (e.g., resolve timing violations).

To address the chip floorplanning problem, we developed an RL method capable of generalizing across chips, meaning that it can learn from experience to become both better and faster at placing

new chips, ushering in a new era in which chip designers are assisted by artificial agents with more experience than any human could ever gain.

Training placement policies that generalize across chips is extremely challenging, as it requires learning to optimize the placement of all possible chip netlists onto all possible canvases. Chip floorplanning is analogous to a game with varying pieces (e.g., netlist topologies, macro counts, macro sizes and aspect ratios), boards (varying canvas sizes and aspect ratios), and win conditions (relative importance of different evaluation metrics or different density and routing congestion constraints). Even one instance of this game (placing a particular netlist onto a particular canvas) has an enormous state-action space. For example, the state space of placing 1000 clusters of nodes on a grid with 1000 cells is on the order of $1000!$ (greater than 10^{2500}), compared to Go, which has a state space of 10^{360} [188].

To enable generalization, we focused on learning transferable representations of chips, grounding representation learning in the supervised task of predicting placement quality. By designing a neural architecture that can accurately predict reward across a wide variety of netlists and their placements, we are able to generate rich feature embeddings of the input netlists. We then use this architecture as the encoder of our policy and value networks to enable transfer learning. In our experiments, we show that, as our agent is exposed to a greater volume and variety of chips, it becomes both faster and better at generating optimized placements for new chip blocks, bringing us closer to a future in which chip designers are assisted by artificial agents with vast chip placement experience.

In addition to the immediate impact on chip floorplanning, the ability of our method to generalize and quickly generate high-quality solutions has major implications, unlocking opportunities for co-optimization with earlier stages of the chip design process. Large-scale architectural explorations were previously impossible, as it took months of human effort to accurately evaluate a given architectural candidate. However, modifying the architectural design can have an outsized impact on performance, and would facilitate full automation of the chip design process. Automating and accelerating the chip design process can also enable co-design of AI and hardware, yielding high performance chips customized to important workloads, such as autonomous vehicles, medical devices, and data centers.

At an abstract level, our method learns to map the nodes of a graph onto a limited set of resources, subject to constraints. Placement optimizations of this form appear in a wide range of science and engineering applications, including hardware design [139], city planning [14], vaccine testing and distribution [142], and cerebral cortex layout [43]. Therefore, we believe that our novel placement optimization methodology can be applied to impactful placement problems beyond chip design.

Beyond the experimental results reported, our method is already having real-world impact and our floorplan solutions have been used in the last four generations of Google’s flagship AI accelerator chip (TPU).

3.3 Related Work

Chip floorplanning is a longstanding challenge, requiring multi-objective optimization over circuits of ever-growing complexity. Since the 1960s, many approaches have been proposed, so far falling into three broad categories: 1) partitioning-based methods, 2) stochastic/hill-climbing methods, and 3) analytic solvers.

Starting in the 1960s, industry and academic labs took a partitioning-based approach to the chip floorplanning problem, proposing [25, 61],[56], as well as resistive-network based methods [40, 204]. These methods are characterized by a divide-and-conquer approach; the netlist and the chip canvas are recursively partitioned until sufficiently small sub-problems emerge, at which point the sub-netlists are placed onto the sub-regions using optimal solvers. Such approaches are quite fast to execute and their hierarchical nature allows them to scale to arbitrarily large netlists. However, by optimizing each sub-problem in isolation, partitioning-based methods sacrifice quality of the global solution, especially routing congestion. Furthermore, a poor early partition may result in an unsalvageable end placement.

In the 1980s, analytic approaches emerged, but were quickly overtaken by stochastic / hill-climbing algorithms, particularly simulated annealing [118, 180, 176]. Simulated annealing (SA) is named for its analogy to metallurgy, in which metals are first heated and then gradually cooled to induce, or anneal, energy-optimal crystalline surfaces. SA applies random perturbations to a given placement (e.g., shifts, swaps, or mirroring of macros), and then measures their effect on the objective function (e.g., half perimeter wirelength). If the perturbation is an improvement, it is applied; if not, it is still applied with some probability, referred to as temperature. Temperature is initialized to a particular value and is then gradually annealed to a lower value. Although SA generates high-quality solutions, it is very slow and difficult to parallelize, thereby failing to scale to the increasingly large and complex circuits of the 1990s and beyond.

The 1990s-2000s were characterized by multi-level partitioning methods [6, 173], as well as the resurgence of analytic techniques [10], such as force-directed methods [138, 87], [158, 191, 211, 210] and non-linear optimizers [97, 98, 96, 37]. The renewed success of quadratic methods was due in part to algorithmic advances, but also to the large size of modern circuits (10-100 million nodes), which justified approximating the placement problem as that of placing nodes with zero area. However, despite the computational efficiency of quadratic methods, they are generally less reliable and produce lower quality solutions than their non-linear counterparts.

Non-linear optimization approximates cost using smooth mathematical functions, such as log-sum-exp [156] and weighted-average [86] models for wirelength, as well as Gaussian [36] and Helmholtz models for density. These functions are then combined into a single objective function using a Lagrange penalty or relaxation. Due to the higher complexity of these models, it is necessary to take a hierarchical approach, placing clusters rather than individual nodes, an approximation which

degrades the quality of the placement.

The last decade has seen the rise of modern analytic techniques, including more advanced quadratic methods [111, 114, 112, 24, 130], and more recently, electrostatics-based methods like ePlace [136], RePlace [48]. Modeling netlist placement as an electrostatic system, ePlace [136] proposed a new formulation of the density penalty where each node (macro or standard cell) of the netlist is analogous to a positively charged particle whose area corresponds to its electric charge. In this setting, nodes repel each other with a force proportional to their charge (area), and the density function and gradient correspond to the system’s potential energy. Variations of this electrostatics-based approach have been proposed to address standard-cell placement [136] and mixed-size placement [135, 137]. RePlace [48] is a recent state-of-the-art mixed-size placement technique that further optimizes ePlace’s density function by introducing a local density function, which tailors the penalty factor for each individual bin size. DREAMPlace [131] further speeds up RePlace by taking a deep learning based approach to optimize the placement and leveraging GPU acceleration. However, the focus of DREAMPlace is standard cell placement optimization, rather than macro placement, and reports comparable quality to RePlace. Therefore, we compare the performance of our method against RePlace.

Modern commercial EDA is dominated by the duopoly consisting of Synopsys (Market Cap: \$72.33B) and Cadence (Market Cap: \$79.87B). These companies sell software for millions of dollars per license per seat, and their licensing agreements prohibit any public comparison against methods. However, as discussed in Section 4.A.1, we performed a blind study in which we compared an early version of AlphaChip against unnamed commercial offerings, and report high-level results. Although neither company publishes or open-sources its methods, we can infer from their press releases and webpages that they employ some manner of reinforcement learning in their chip design products [195, 171, 144, 29], following publication of AlphaChip work in *Nature*.

There are numerous opportunities for machine learning to advance physical design [100, 101, 7]. Recent work [88] proposes training a model to predict the number of Design Rule Check (DRC) violations for a given macro placement. DRCs are rules that ensure that the placed and routed netlist adheres to tape-out requirements. To generate macro placements with fewer DRCs, [88] use the predictions from this trained model as the evaluation function in simulated annealing. While this work represents an interesting direction, it reports results on netlists with no more than 6 macros, far fewer than any modern block, and does not consider the effect of place and route optimizations, which can dramatically alter the number of DRCs. Furthermore, although adhering to the DRC criteria is a necessary condition, the primary objective of macro placement is to optimize for wirelength, timing (e.g., Worst Negative Slack (WNS) and Total Negative Slack (TNS)), power, and area, and this work does not even consider these metrics.

To address this classic problem, we propose a new category of approach: an end-to-end learning-based method. This new approach is most closely related to analytic solvers, particularly non-linear

ones, in that all of these methods optimize an objective function via gradient updates. However, our approach differs from prior approaches in its ability to learn from past experience to generate higher-quality placements on new chips. Unlike existing methods that optimize the placement for each new chip from scratch, our work leverages knowledge gained from placing prior chips to become better over time. In addition, our method enables direct optimization of the target metrics, such as wirelength, density, and congestion, without having to define convex approximations of those functions as is done in other approaches [48, 136]. Not only does our formulation make it easy to incorporate new cost functions as they become available, but it also allows us to weight their relative importance according to the needs of a given chip block (e.g., timing-critical or power-constrained).

Domain adaptation is the problem of training policies that can learn across multiple experiences and transfer the acquired knowledge to perform better on new unseen examples. In the context of chip floorplanning, domain adaptation involves training a policy across a set of chip netlists and then applying that trained policy to a new unseen netlist. The use of deep learning for combinatorial optimization is an area of growing interest, including approaches to the Traveling Salesman Problem [22], [110], Neural Architecture Search [242], and Model Parallelism [151, 148]. More recently, there has been work on domain adaptation for compiler optimization [3, 239, 165, 240] and the Maximum Cut problem [19]. Our approach not only leverages past experience to reduce training time, but also produces higher quality results when exposed to more instances of the problems. To our knowledge, our method is the first deep RL approach used in production to solve a combinatorial optimization problem, namely in the design of the last four generations of Google TPU.

3.4 Chip floorplanning as a learning problem

The underlying problem is a high-dimensional contextual bandits problem [122], but similar to prior work, such as [207, 22, 151, 148], we have chosen to reformulate it as a sequential Markov Decision Process (MDP), as this allows us to more easily incorporate the problem constraints as described below. Our MDP consists of four key elements:

- **states** encode information about the partial placement, including the netlist (adjacency matrix), node features (width, height, type), edge features (number of connections), current node (macro) to be placed, and metadata of the netlist graph (routing allocations, total number of wires, macros, and standard cell clusters).
- **actions** are all possible locations (grid cells of the chip canvas) onto which the current macro can be placed without violating any hard constraints on density or blockages.
- **state transitions** define the probability distribution over next states, given a state and an action.

- **rewards** are 0 for all actions except the last action where the reward is a negative weighted sum of proxy wirelength, congestion, and density as described below.

We train a policy (an RL agent) modeled by a neural network that through repeated episodes (sequences of states, actions, and rewards), learns to take actions that will maximize cumulative reward (see Figure 3.3). We use Proximal Policy Optimization (PPO) [179] to update the parameters of the policy network, given the cumulative reward for each placement.

We can formally define the objective function as follows:

$$J(\theta, G) = \frac{1}{K} \sum_{g \sim G} E_{g, p \sim \pi_\theta} [R_{p,g}] \quad (3.1)$$

Here $J(\theta, G)$ is the cost function. The agent is parameterized by θ . The dataset of netlists of size K is denoted by G with each individual netlist in the dataset written as g . $R_{p,g}$ is the episode reward of a placement p drawn from the policy network applied to netlist g .

$$R_{p,g} = -\text{Wirelength}(p, g) - \lambda \text{Congestion}(p, g) - \gamma \text{Density}(p, g) \quad (3.2)$$

In each iteration, the RL agent (policy network) sequentially places the macros. Once all macros are placed, we use a force-directed method [158, 191, 211, 210] to approximately place clusters of standard cells. The reward at the end of each iteration is calculated as a linear combination of the approximate wirelength, congestion, and density (Equation 3.2). In our experiments, congestion weight λ is set to 0.01, density weight γ is set to 0.01, and the max density threshold is set to 0.6.

3.5 Designing Domain-Adaptive Policies

As mentioned earlier, developing domain-adaptive policies for the chip floorplanning problem is extremely challenging, as this problem is analogous to a game with varying pieces, boards, and win conditions, and has an enormous state-action space. To address this challenge, we first focused on learning rich representations of the state space. Our intuition was that a policy capable of the general task of chip placement should also be able to encode the state associated with a new unseen chip into a meaningful signal at inference time. We therefore trained a neural network architecture capable of predicting reward on placements of new netlists, with the ultimate goal of using this architecture as the encoder layer of our policy.

To train this supervised model, we needed a large dataset of chip placements and their corresponding reward labels. We therefore created a dataset of 10,000 chip placements where the input is the state associated with a given placement and the label is the reward for that placement.

To accurately predict the reward labels and generalize to unseen data, we developed a novel edge-based graph neural network architecture, which we call Edge-GNN (Edge-Based Graph Neural Network). The role of this network is to embed the netlist, distilling information about the type and connectivity of nodes into a low-dimensional vector representation which can be used in downstream tasks. To show the impact of our edge-based neural architecture on generalization, please see Figure 3.1.



Figure 3.1: The zero-shot performance of our Edge-GNN vs. GCN [116]. The agent with an Edge-GNN architecture is more robust to over-fitting and yields higher quality results, as measured by average zero-shot performance, on the test blocks in Table 3.4.

In Edge-GNN, we create an initial representation of each node by concatenating its features, including node type, width, height, x and y coordinates, and its connectivity to other nodes. We then iteratively perform the following updates: 1) each edge updates its representation by applying a fully connected network to a concatenation of the two nodes which it connects, and 2) each node updates its representation by passing the mean of all in/outgoing edges into another fully connected network. The node and edge updates are shown in Equation 3.3.

$$\begin{aligned} e_{ij} &= f_{c_e}(\text{concat}(v_i | v_j | w_{ij}^e)) \\ v_i &= f_{c_v}(\text{mean}_{j \in \mathcal{N}(v_i)}(e_{ij})) \end{aligned} \quad (3.3)$$

Node embeddings are denoted by v_i for $1 \leq i \leq N$, where N is the total number of macros and standard cell clusters. Vector representations of edges connecting nodes v_i and v_j are denoted as e_{ij} . The outputs of the algorithm are the node and edge embeddings.

The supervised model is trained via regression to minimize the weighted sum of mean squared loss

(negative reward). This supervised task allowed us to find the features and architecture necessary to generalize reward prediction across netlists. To incorporate Edge-GNN into our RL policy network, we removed the prediction layer and then used it as the encoder of the policy network as shown in Figure 3.2.

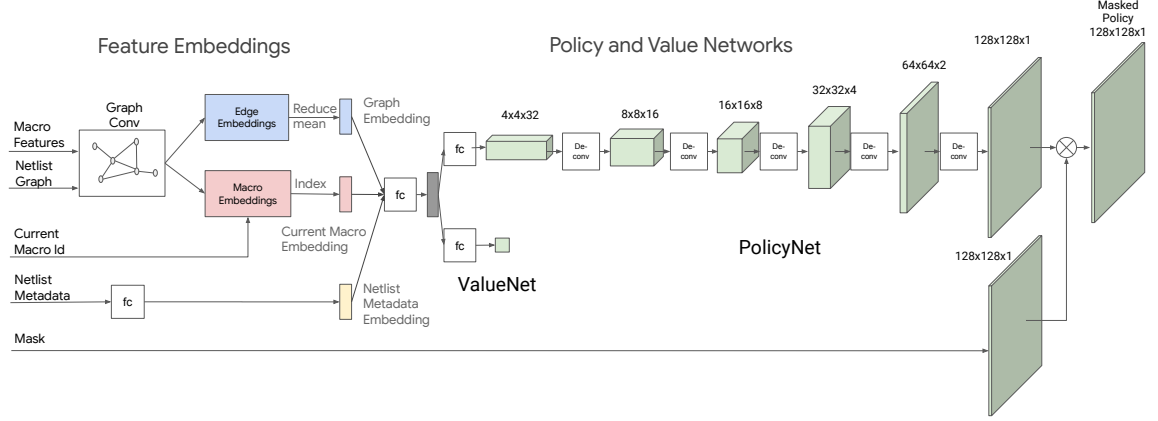


Figure 3.2: Policy and value network architecture. An embedding layer encodes information about the netlist adjacency, node features, and the current macro to be placed. The policy and value networks then output a probability distribution over available grid cells and an estimate of the expected reward for the current placement, respectively.

To place a new netlist at inference time, we load the pre-trained weights of the policy network and apply it to the new netlist. We refer to placements generated by a pre-trained policy with no finetuning as zero-shot placements. Such a placement can be generated in subseconds, because it requires only a single forward pass through the pre-trained policy for each macro. We can further optimize placement quality by finetuning the policy network. Doing so gives us the flexibility to either use the pre-trained weights (which have learned a rich representation of the input state) or further finetune these weights to optimize for the properties of a particular chip netlist.

Figure 3.2 shows an overview of the proposed policy network (modeled by π_θ in Equation 3.1) and value network architectures. The input is the chip netlist (represented as an adjacency matrix and list of node features), id of the current node to be placed, metadata of the netlist and process technology node (e.g., 7nm). The netlist is fed into our edge-GNN architecture to generate embeddings of the partially placed netlist and of the current node. We use a feedforward network to embed the metadata. These embedding vectors are then concatenated to form the state embedding, which is passed to another feedforward neural network to generate a final representation of the state. This state is then fed into the policy network (composed of 5 deconvolutions¹, batch normalization [90], and ReLU activation layers [155]) to generate a probability distribution over actions and into a value network (composed of a feedforward network) to predict the value of the input state.

¹The deconvolution layers have kernel size of 3x3, stride of 2, and 16, 8, 4, 2, and 1 filter channels, respectively.

3.6 Methods

In the following, we provide details of the proposed methodologies.

3.6.1 Problem Statement

In this chapter, we target the chip floorplanning problem, in which the objective is to map the nodes of a netlist (a hypergraph encoding chip components and their connectivity via wires) onto a chip canvas (a bounded 2D space), such that final power, performance, and area (PPA) is optimized. In this section, we describe an overview of how we formulate this task as an RL problem, followed by a detailed description of the reward function, action and state representations, policy architecture, and policy updates.

3.6.2 Overview of Our Approach

We take a deep reinforcement learning approach to the chip floorplanning problem, where an RL agent (policy network) sequentially places the macros. Once all macros are placed, we use a force-directed method [158, 191, 211, 210] to place clusters of standard cells, as shown in Figure 3.3.

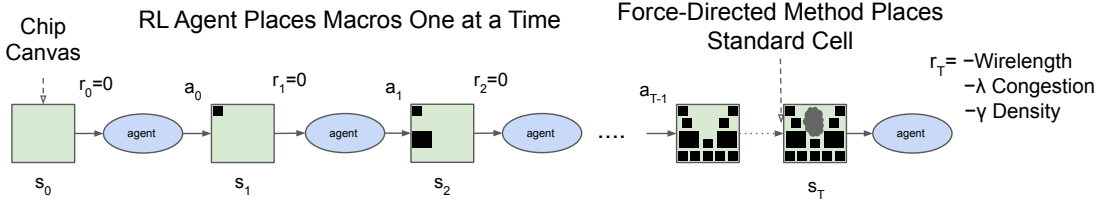


Figure 3.3: Overview of our method and training regimen. In each training iteration, the RL agent places macros one at a time (actions, states, and rewards are denoted by a_i , s_i , and r_i). Once all macros are placed, the standard cells are placed using a force-directed method. The intermediate rewards are zero. The reward at the end of each iteration is calculated as a linear combination of the approximate wirelength, congestion, and density, and is provided as feedback to the agent to optimize its parameters for the next iteration.

In our setting, at the initial state, s_0 , we have an empty chip canvas and an unplaced netlist. At each step, one macro is placed, and the final state s_T corresponds to a completely placed netlist. Thus, T is equal to the total number of macros in the netlist. At each time step t , the agent begins in state (s_t), takes an action (a_t), arrives at a new state (s_{t+1}), and receives a reward (r_t) from the environment (0 for $t < T$ and negative proxy cost for $t = T$).

We define s_t to be a concatenation of features representing the state at time t , including a graph embedding of the netlist (including both placed and unplaced nodes), a node embedding of the current macro to place, metadata about the netlist, and a mask representing the feasibility of placing the current node onto each cell of the grid.

The action space is all valid placements of the t^{th} macro, which is a function of the density mask. Action a_t is the cell placement of the t^{th} macro predicted by the RL policy network.

s_{t+1} is the next state, which includes an updated representation containing information about the newly placed macro, an updated density mask, and an embedding for the next node to be placed.

In our formulation, r_t is 0 for every time step except for the final r_T , where it is a weighted sum of approximate wirelength, congestion, and density.

Through repeated episodes (sequences of states, actions, and rewards), the policy network learns to take actions that will maximize cumulative reward. We use Proximal Policy Optimization (PPO) [179] to update the parameters of the policy network, given the cumulative reward for each placement.

In this section, we define the reward r , state s , actions a , policy network architecture $\pi_\theta(a|s)$ parameterized by θ , and finally the optimization method we use to train those parameters.

3.6.3 Detailed Methodology

Our goal in this chapter is to minimize power, performance and area, subject to constraints on routing congestion and density. Our true reward is the output of a commercial EDA tool, including wirelength, routing congestion, density, power, timing, and area. However, RL policies require 10,000s of examples to learn effectively, so it is critical that the reward function be fast to evaluate, ideally running in a few milliseconds. In order to be effective, these approximate reward functions must also be positively correlated with the true reward. Therefore, a component of our cost is wirelength, because it is not only much cheaper to evaluate, but also correlates with power and performance (timing).

To combine multiple objectives into a single reward function that can be optimized, we take the weighted sum of proxy wirelength, congestion, and density, where the weights can be used to explore the trade-off between these metrics.

While we treat congestion as a soft constraint (i.e., lower congestion improves the reward function), we treat density as a hard constraint, masking out actions (grid cells to place nodes onto) whose density exceeds the target density.

To keep the runtime per iteration small, we apply several approximations to the calculation of the reward function:

1. We group millions of standard cells into a few thousand clusters using hMETIS [107], a partitioning technique based on the minimum cut objective. Once all macros are placed, we use a force-directed method to place the standard cell clusters. Doing so enables us to generate an approximate but fast standard cell placement that facilitates policy network optimization.
2. We discretize the grid to a few thousand grid cells and place the center of macros and standard cell clusters onto the center of the grid cells.

3. When calculating wirelength, we make the simplifying assumption that all wires leaving a standard cell cluster originate at the center of the cluster.
4. To calculate routing congestion cost, we only consider the average congestion of the top 10% most congested grid cells.

A chip netlist typically consists of hundreds of macros and millions of standard cells. Due to their negligible area, standard cells can be approximated as points with zero area, allowing for analytic solvers to optimally place them with a small margin of error. Macros, on the other hand, have much larger area and cannot be optimally placed with these same analytic techniques. We chose to target macro placement as it is a much more challenging problem that previously required human experts to iterate for months to generate a high-quality placement.

3.6.4 Synthesis of the input netlist

We use a commercial tool to synthesize the netlist from RTL. Synthesis is physical-aware in the sense that it has access to the floorplan size and the locations of the Input-Output (I/O) pins, which were informed by inter and intra-block level information.

3.6.5 Selection of grid rows and columns

Given the dimensions of the chip canvas, there are many choices to discretize the 2D canvas into grid cells. This decision impacts the difficulty of optimization and the quality of the final placement. We limit the maximum number of rows and columns to 128. We treat choosing the optimal number of rows and columns as a bin-packing problem and rank different combinations of rows and columns by the amount of wasted space they incur. We use an average of 30 rows and columns in our experiments.

3.6.6 Selection of macro order

To select the order in which the macros are placed, we sort macros by descending size and break ties using a topological sort. By placing larger macros first, we reduce the chance of there being no feasible placement for a later macro. The topological sort can help the policy network learn to place connected nodes close to one another. Another potential approach would be to learn to jointly optimize the ordering of macros and their placement, making the choice of which node to place next part of the action space. However, this enlarged action space would significantly increase the complexity of the problem, and we found that this heuristic worked in practice.

3.6.7 Clustering of Standard Cells

In order to quickly place standard cells to provide signal to our RL policy, we first cluster millions of standard cells into a few thousand clusters. There has been a large body of work on clustering for chip netlists [8, 31, 34, 9, 64, 63]. As has been suggested in the literature [99], such clustering helps not only with reducing problem size, but also helps “prevent mistakes” (e.g., prevents timing paths from being split apart). We also provide the clustered netlist to each of the baseline methods with which we compare. To perform this clustering, we employed a standard open-source library, hMETIS [107], which is based on multilevel hypergraph partitioning schemes with two important phases: 1) Coarsening phase, and 2) uncoarsening and refinement phase. After clustering with hMETIS, we rebalance cluster sizes using heuristics based on an initial placement from physical synthesis, the previous step in the chip design process².

3.6.8 Generation of Adjacency Matrix

In order to convert the netlist hypergraph into an adjacency matrix that can be consumed by the Edge-GNN encoder, we apply the transformation described below. Technically, the input netlist is a hypergraph, as its edges (wires) may connect more than two chip components. However, for the purposes of the placement problem, this distinction is uninteresting and we reduce the hypergraph to a graph in which each edge connects just two components (either standard cells or macros), facilitating the use of graph neural network based methods.

For each pair of nodes in the clustered netlist (either macros or clusters of standard cells), we generate an edge in the adjacency matrix with the following weight. If the register distance between the two nodes is greater than 4, then no edge is created. Otherwise, we apply an exponentially decaying weight as the distance grows, starting with 1 if the distance is 0 and halved with each additional unit of distance.

3.6.9 Placement of Standard Cells

To place standard cell clusters, we use an approach similar to classic force-directed methods[183]. We represent the netlist as a system of springs that apply force to each node, according to the $weight \times distance$ formula, causing tightly connected nodes to be attracted to one another. We also introduce a repulsive force between overlapping nodes to reduce placement density. After applying all forces, we move nodes in the direction of their force vector. To reduce oscillations, we set a maximum distance for each move.

²Note that this cluster rebalancing step has no apparent impact on performance (see ablation study in Table 4.A.1), and we recommend instead instructing hMETIS to produce balanced clusters from the outset by setting UBFactor to 1.

3.6.10 Postprocessing

To prepare the placements for evaluation by a commercial EDA tool, we perform a simple legalization step to snap macros to the nearest power grid. We then fix the macro placements and use an EDA tool to place the standard cells and evaluate the placement.

3.6.11 Reward

Wirelength

Following the literature [183, 30, 104, 103], we employ half perimeter wirelength (HPWL), the most commonly used approximation for wirelength. HPWL is defined as the half perimeter of the bounding boxes for all nodes in the netlist. The HPWL for a given net (edge) i is shown in the equation below:

$$HPWL(i) = (MAX_{b \in i}\{x_b\} - MIN_{b \in i}\{x_b\} + 1) + (MAX_{b \in i}\{y_b\} - MIN_{b \in i}\{y_b\} + 1) \quad (3.4)$$

Here x_b and y_b show the x and y coordinates of the end points of net i . The overall HPWL cost is then calculated by taking the normalized sum of all half-perimeter bounding boxes, as shown in Equation 3.5. Here $q(i)$ is a normalization factor which improves the accuracy of the estimate by increasing the wirelength cost as the number of nodes increases, where $N_{netlist}$ is the number of nets. We calculate the total HPWL as follows:

$$HPWL(netlist) = \sum_{i=1}^{N_{netlist}} q(i) \times HPWL(i) \quad (3.5)$$

Wirelength also has the advantage of correlating with other important metrics, such as power and timing. Although our method does not optimize directly for these other metrics, it generates placements that meet design criteria with respect to power and timing (as shown in Table 3.4).

Routing congestion

We also followed convention in calculating proxy congestion [114], using a simple deterministic routing based on the locations of the driver and loads on the net. The routed net occupies a certain portion of available routing resources (determined by the underlying semiconductor fabrication technology) for each grid cell through which it passes. We keep track of vertical and horizontal allocations in each grid cell separately. To smooth the congestion estimate, we run 5×1 convolutional filters in both the vertical and horizontal direction. After all nets are routed, we take the average of the top 10% congestion values, drawing inspiration from the ABA10 metric in MAPLE [114]. The congestion

cost in Equation 3.2 is the top 10% average congestion calculated by this process.

Density

We treat density as a hard constraint, disallowing the policy network from placing macros in locations which would cause density to exceed the target ($max_{density}$) or which would result in infeasible macro overlap. This approach has two benefits: (1) it reduces the number of invalid placements generated by the policy network, and (2) it reduces the search space of the optimization problem, making it more computationally tractable.

A feasible placement of a standard cell cluster must meet the following criterion: the density of placed items in each grid cell must not exceed a given target density threshold ($max_{density}$). We set this threshold to be 0.6 in our experiments to avoid over-utilisation, which would render placements unusable. To meet this constraint, during each RL step, we calculate the current density mask, a binary $m \times n$ matrix representing grid cells onto which we can place the center of the current node without violating the density threshold. Before selecting an action, we first take the dot product of the mask and the policy network output and then sample from the resulting probability distribution over feasible locations. This approach prevents the policy network from generating placements with overlapping macros or dense standard cell areas.

We also enable blockage-aware placements (such as clock straps) by setting the density function of the blocked areas to 1.

3.6.12 Action Representation

For policy optimization purposes, we convert the canvas into an $m \times n$ grid. Thus, for any given state, the action space (or the output of the policy network) is the probability distribution of placements of the current macro over the $m \times n$ grid. The action is then sampled from this probability distribution.

3.6.13 State Representation

Our state contains information about the adjacency matrix corresponding to the clustered netlist, its node features (width, height, type), edge features (number of connections), current node (macro) to be placed, and metadata of the netlist and the underlying technology (e.g., routing allocations, total number of wires, macros, and standard cell clusters). Next, we discuss how we process these features to learn effective representations for the chip floorplanning problem.

3.6.14 Enabling Transfer Learning

In order to discover domain-adaptive architectures, we propose grounding the policy architecture search in the supervised task of predicting the value of reward functions. We take this approach

because exploration would be far costlier in an RL setting, and the underlying complexity of training a domain-adaptive policy network would be prohibitively high, as it involves an immense state-space encompassing all possible placements of all possible chips. Furthermore, different netlists and grid sizes can have very different properties, including differing numbers of nodes, macro sizes, netlist topologies, and canvas widths and heights.

The intuition behind this approach is that a policy network architecture capable of transferring placement optimization across chips should also be able to encode the state associated with a new unseen chip into a meaningful signal at inference time. We therefore propose training a neural network architecture capable of predicting reward on new netlists, with the ultimate goal of using this architecture as the encoder layer of our policy network.

To train this supervised model, we needed a large dataset of chip floorplans and their corresponding reward labels. We therefore created a dataset of 10,000 chip floorplans where the input is the state associated with a given floorplan and the label is the reward for that floorplan (wirelength and congestion). We built this dataset by generating 2,000 floorplans for each of 5 TPU blocks. To collect diverse floorplans, we trained a vanilla policy network with various congestion weights (ranging from 0 to 1) and random seeds, and collected snapshots of floorplans throughout the course of policy training. An untrained policy network starts off with random weights and the generated floorplans are of low quality, but as the policy network trains, the quality of generated floorplans improves, allowing us to gather a diverse dataset with floorplans of varying quality.

To train a supervised model capable of accurately predicting wirelength and congestion labels and generalizing to unseen data, we developed a novel graph neural network architecture (Edge-GNN) to embeds information about the netlist. The role of the Edge-GNN is to distill information about the type and connectivity of a node into a low-dimensional vector representation which can be used in downstream tasks. Some examples of such downstream tasks are node classification [157], device placement [239], link prediction [237], and Design Rule Violations (DRCs) prediction [224].

We create a vector representation of each node by first concatenating its features, including node type, width, height, and x and y coordinates. We also pass node adjacency information as input to our algorithm. We then repeatedly perform the following updates: 1) each edge updates its representation by applying a fully connected network to an aggregated representation of intermediate node embeddings, and 2) each node updates its representation by taking the mean of adjacent edge embeddings. The node and edge updates are shown in Equation 3.3.

Node embeddings are denoted by v_i s for $1 \leq i \leq N$, where N is the total number of macros and standard cell clusters. The vector representations of the edge connecting nodes v_i and v_j is represented as e_{ij} . Both edge (e_{ij}) and node (v_i) embedding are 32-dimensional. v_i is initialized by passing the node features (type, width, height, x, y) through a feedforward network. fc_e is a 65×32 feedforward network and w_{ij}^e s are 1×1 weights corresponding to the number of nets between

adjacent nodes. $\mathcal{N}(v_i)$ shows the neighbors of v_i . The outputs of the algorithm are the node and edge embeddings.

Our supervised model consists of: (1) The graph neural network (Edge-GNN) described above which embeds information about node type and the netlist adjacency matrix. (2) A fully connected feedforward network that embeds netlist metadata, including information about the underlying semiconductor technology (horizontal and vertical routing capacity), the total number of nets (edges), macros, and standard cell clusters, canvas size, and number of rows and columns in the grid. (3) A fully connected feedforward network (the prediction layer) whose input is a concatenation of the netlist adjacency matrix and metadata embeddings and whose output is the reward prediction. The netlist embedding is created by applying a reduce mean function on the edge embeddings. The supervised model is trained via regression to minimize the weighted sum of the mean squared loss of wirelength and congestion.

This supervised task allowed us to find the features and architecture necessary to generalize reward prediction across netlists. To incorporate this architecture into our policy network, we simply removed the prediction layer and then used the remaining network as the encoder of the policy network as shown in Figure 3.2.

To handle different grid sizes corresponding to different choices of rows and columns, we set the grid size to 128×128 , and mask the unused L-shaped section for grid sizes smaller than 128 rows and columns.

To place a new test netlist at inference time, we load the pre-trained weights of the policy network and apply it to the new netlist. We refer to placements generated by a pre-trained policy network with no finetuning as zero-shot placements. Such a placement can be generated in less than a second, because it only requires a single inference step of the pre-trained policy network for each macro. We can further optimize placement quality by finetuning the policy network, meaning that we have the option to either use the pre-trained weights (which have learned a rich representation of the input state) directly at inference or further finetune these weights to optimize for the properties of a particular chip netlist.

3.6.15 Policy Network Architecture

Figure 3.2 depicts an overview of the policy network (modeled by π_θ in Equation 3.1) and the value network architecture that we developed for chip floorplanning. The input to these networks is the adjacency matrix and node features corresponding to the chip netlist, the id of the current node to be placed, and the metadata of the netlist and the semiconductor technology. The netlist is passed through our graph neural network architecture (Edge-GNN) as described earlier. Edge-GNN generates embeddings of (1) the partially placed netlist and (2) the current node. We use a simple feedforward network to embed (3) the metadata. These three embedding vectors are then concatenated to form

the state embedding, which is passed to a feedforward neural network. The output of the feedforward network is then fed into the policy network (composed of 5 deconvolutions³, Batch Normalization, and ReLU activation layers) to generate a probability distribution over actions and passed to a value network (composed of a feedforward network) to predict the value of the input state.

3.6.16 Policy Network Update: Training Parameters

In Equation 3.1, the objective is to train a policy network π_θ that maximizes the expected value (E) of the reward ($R_{p,g}$) over the policy network’s placement distribution. To optimize the parameters of the policy network, we use Proximal Policy Optimization (PPO) [179] with a clipped objective as shown below:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where \hat{E}_t represents the expected value at timestep t , r_t is the ratio of the new policy and the old policy, and \hat{A}_t is the estimated advantage at timestep t .

See Appendix 3.A for an early exploration of other RL optimization algorithms for this task.

3.7 Empirical Evaluation

In this section, we evaluate the ability of our method to generalize, explore the impact of using pre-trained policies, and compare our method to state-of-the-art baselines. We also inspect the visual appearance of generated placements and provide insights into the behavior of our policy.

In terms of resource usage, for pre-training, we used the same number of workers as blocks in the training dataset (e.g., for the largest training set with 20 blocks, we pre-trained with 20 workers) and the pre-training runtime was 48 hours. To generate the fine-tuning results in Table 3.4, our method ran on 16 workers for up to 6 hours, but the runtime was often significantly lower due to early stopping. For both pre-training and finetuning, a worker consists of an Nvidia Volta GPU and 10 CPUs each with 2GB of RAM. For zero-shot mode (applying a pre-trained policy to a new netlist with no fine-tuning), we can generate a placement in less than a second on a single GPU.

3.7.1 Experimental Setup

In order to perform a fair comparison, we ensured that our method and all baseline methods had access to the same inputs and the same evaluation settings. Figure 3.4 shows the flow that we used to conduct the evaluations.

³The deconvolutions layers have a 3x3 kernel size with stride 2 and 16, 8, 4, 2, and 1 filter channels respectively.

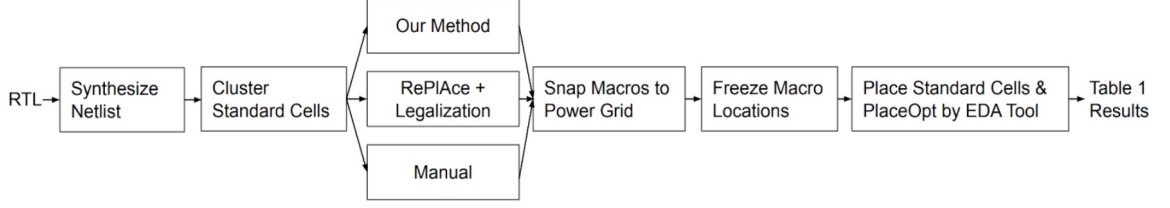


Figure 3.4: Evaluation workflow for producing the results in Table 3.4. We allow each method access to the same clustered netlist. We use the same hyperparameters (to the extent possible) in all the methods. Once the placement is completed by each method (this includes the legalization step for RePIAce), we snap the macros to the power grids, freeze the macro locations, and use a commercial EDA tool to place the standard cells and report final results.

Once each method finishes placing the netlist, the macro locations are frozen and snapped to the power grid. Next, the EDA tool performs standard cell placement. The settings for the EDA tool are drawn directly from our production flow and thus we cannot share all details. The final metrics in Table 3.4, are reported after PlaceOpt, meaning that global routing has been performed by the EDA tool.

Clustering standard cells allowed our method to more effectively optimize the placement of macros. We therefore gave RePIAce access to clustered standard cells and found that its performance also improved, so we reported results of RePIAce on the netlist with clustered standard cells. Although RePIAce has a default density threshold of 1.0, we found that our setting of 0.6 resulted in better performance, so that is what we used to report RePIAce performance. In all other cases, we used the default settings and cost functions for RePIAce.

For reproducibility, we provide all architectural details and hyperparameter settings for our RL algorithm in Table 3.1, as well as for the FD method used to place standard cells in Table 3.2.

To cluster the standard cells for each chip block, we used hMETIS [107], which partitions millions of standard cells into thousands of clusters. The hyperparameters for hMETIS are listed in Table 3.3. For all other hMETIS hyperparameters, we simply use the default settings. Please refer to the hMETIS manual [1] for the value of these defaults and for more detailed information about each hyperparameter. Note that we use a licensed version of hMETIS, but to our knowledge, the same features are available in the open-source version.

To avoid overfitting, we employ an early stopping mechanism that halts RL training once the policy converges. More precisely, training stops when it has been two hours since the evaluation return improved by at least 0.5% over the best return so far.

3.7.2 Open-Source Benchmark: Ariane RISC-V

For the Ariane benchmark, we used the following open-source design [234] (<https://github.com/pulp-platform/ariane>), and mapped all logical memories to physical memories of size 256x16,

Hyperparameter	Value
Learning rate	1.00E-04
Optimizer	Adam
Num Epochs	4
Num GPUs	16
Num CPUs per GPU worker	10
Batch size per GPU	64
Effective batch size (Num GPUS \times Batch size per GPU)	1024
Clip Grad	1
Num actors per GPU	32
Weight initializer	Xavier
Episodes per rollout	2
PPO loss:	
Clipping parameter (ϵ)	0.2
Value Coeff	0.5
Entropy Coeff	0.01
Discount (γ)	1

Table 3.1: Hyperparameters used for finetuning the RL agent. The pre-training hyperparameters are the same, except for the number of GPUs and the effective batch size. For pre-training, we use one GPU per block in the training dataset (our largest dataset has 20 blocks).

Hyperparameter	Value	Description
Number of schedules	3	Number of schedules to run the force-directed algorithm
Steps	[100, 100, 100]	Number of steps of the force-directed algorithm during each schedule
Move Distance Factors	[1.0, 1.0, 1.0]	Maximum distance relative to canvas size that a node can move in a single step of the force-directed algorithm
Attract Factors	[100, 1e-3, 1e-5]	The spring constants between two connected nodes in the force-directed algorithm
Repel Factors	[0, 1e6, 1e7]	The repellent factor for spreading the nodes to avoid congestion in the force-directed algorithm
I/O Factors	[1.0, 1.0, 1.0]	The I/O factor for multiplying the forces from the I/O ports to the nodes in the force-directed algorithm

Table 3.2: Hyperparameters used for the force-directed algorithm that places standard cell clusters.

Name	Val	Definition
UBfactor	5	The extent to which unbalanced partitions are permitted.
Nruns	10	The number of bisections performed by hMETIS, the best of which is returned as the final solution.
CType	5	Edge Coarsening (EC) algorithm (heavy-edge maximal matching). In this mode, pairs of vertices are grouped together if they are connected by multiple hyperedges.
RType	3	Early-Exit Fiduccia-Mattheyses refinement scheme (FM-EE) algorithm. In this mode, the FM iteration is aborted if the quality of the solution does not improve after a relatively small number of vertex moves.
Vcycle	3	Performs Vcycle refinement on each intermediate solution, meaning that each one of the Nruns bisections is also refined using Vcycles.

Table 3.3: Hyperparameters used to generate standard cell clusters with hMETIS [107].

resulting in 133 macros. In Figure 3.9, we compare a placement generated by our method trained from scratch and one that was generated in zero-shot mode by a pre-trained policy.

3.7.3 Google TPU Results: Comparing with Baseline Methods

In this section, we compare our method with the state-of-the-art RePlAce [48] and to the production design of the previous generation of TPU (TPU-v4), which was generated by a team of human physical designers. The results are shown in Table 3.4.

To perform a fair comparison, we ensured that all methods had the same experimental setup, including the same inputs and the same EDA tool settings. Note that we ran all of the evaluations of RePlAce and our method ourselves, but we relied on the TPU physical design team to share metrics for their best performing manual placements, and they may have evaluated with a slightly different EDA version. For more details, please see Figure 3.4.

For our method, we use a policy pre-trained on the largest dataset (20 TPU blocks) and then finetune it on 5 target unseen blocks (denoted by Blocks 1 to 5) for no more than six hours. Due to confidentiality, we cannot disclose the details of these blocks, but each contains up to 131 macros and millions of standard cells.

When evaluating the quality of a chip floorplan, there are several metrics that are important and which trade off against each other. There is no single metric that can be used to capture the overall quality of a placement, so we report all key metrics, including total wirelength, timing, routing congestion (horizontal and vertical), area, and power. Timing is reported via TNS (total negative

Name	Method	Timing		Area Total (μm^2)	Power Total (W)	Wirelength (m)	Congestion	
		WNS (ps)	TNS (ns)				H (%)	V (%)
Block 1	RePLAce	374	233.7	1693139	3.70	52.14	1.82	0.06
	Manual	136	47.6	1680790	3.74	51.12	0.13	0.03
	Ours	84	23.3	1681767	3.59	51.29	0.34	0.03
Block 2	RePLAce	97	6.6	785655	3.52	61.07	1.58	0.06
	Manual	75	98.1	830470	3.56	62.92	0.23	0.04
	Ours	59	170	694757	3.13	59.11	0.45	0.03
Block 3	RePLAce	193	3.9	867390	1.36	18.84	0.19	0.05
	Manual	18	0.2	869779	1.42	20.74	0.22	0.07
	Ours	11	2.2	868101	1.38	20.80	0.04	0.04
Block 4	RePLAce	58	11.2	944211	2.21	27.37	0.03	0.03
	Manual	58	17.9	947766	2.17	29.16	0.00	0.01
	Ours	52	0.7	942867	2.21	28.50	0.03	0.02
Block 5	RePLAce	156	254.6	1477283	3.24	31.83	0.04	0.03
	Manual	107	97.2	1480881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1472302	3.28	36.59	0.01	0.03

Table 3.4: Comparisons against baselines. Here, we compare our method with the state-of-the-art (RePLAce [48]) method and manual placements using an industry standard electronic design automation (EDA) tool. For all metrics in this table, lower is better.

slack) and WNS (worst negative slack). Negative slack is a measure of the extent to which the latency of the signal exceeds the expected latency. Timing and congestion are constraints, whereas wirelength, power, and area are metrics to optimize.

To compare with RePLAce which has a different objective function, we treat the output of a commercial EDA tool as ground truth. To perform this comparison, we fix the macro placements generated by our method and by RePLAce and allow a commercial EDA tool to further optimize the standard cell placements with settings drawn from our production workflow. We used the version of RePLAce provided in [2], based on the commit on January 9th, 2020. Except for density threshold (where RePLAce benefited from a lower threshold than its default), we used the default settings and did not use the timing-driven capability of RePLAce.

As shown in Table 3.4, our method outperforms RePLAce in generating placements that meet design criteria. While RePLAce is faster and runs in under an hour on a single Intel CPU 3.7 GHz, the placements are generally of lower quality. Given constraints imposed by the underlying process technology node, placements will not be able to meet timing constraints in the later stages of the design flow if WNS is significantly above 150 ps or if the horizontal or vertical congestion is over 1%, rendering many RePLAce placements (Blocks 1, 2, 3) unusable. These results demonstrate that our approach is effective in generating high-quality placements that meet design criteria.

Table 3.4 also shows the results for the manual baseline, which is the actual production design of the previous TPU chip. This baseline was generated by the TPU’s physical design team, and involved many iterations of placement optimization, guided by feedback from a commercial EDA tool over a period of several months. Both our method and human experts consistently generate viable placements that meet timing and congestion requirements. However, our method also outperforms or matches

manual placements in area, power, and wirelength. Furthermore, our end-to-end learning-based approach takes far less time to meet design criteria.

3.7.4 Domain Adaptation Results

Figure 3.5 compares the quality of placements generated using pre-trained policies to those generated by training the policy from scratch. The training dataset is composed of blocks of TPU and of the open source Ariane RISC-V CPU [234]. In each experiment, we pre-train the policy on all blocks except for the target block on which we evaluate. We show results for zero-shot mode, as well as after finetuning the pre-trained policy on a particular design for 2 and 12 hours.

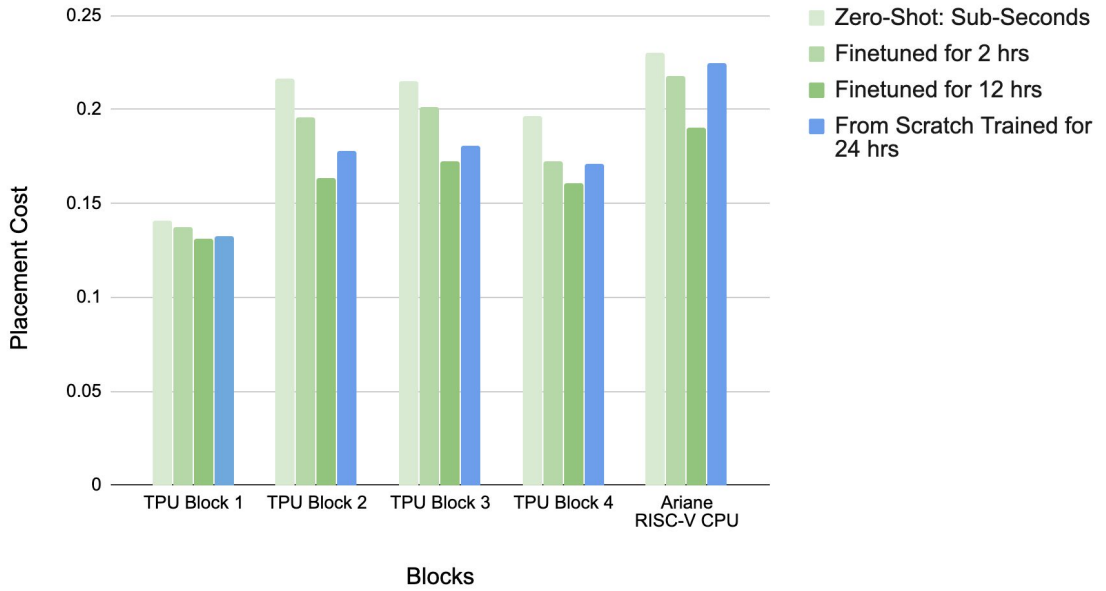


Figure 3.5: Training from scratch vs. fine-tuning for varying amounts of time. For each block, we show zero-shot results, results after finetuning for 2 and 12 hours, and results for policies trained from scratch. As can be seen in the table, the pre-trained policy network consistently outperforms the policy network trained from scratch, demonstrating the effectiveness of learning from training data offline.

The policy trained from scratch takes much longer to converge, and even after 24 hours, the results (as evaluated by the reward function) are worse than what the finetuned policy achieves in 12 hours. This demonstrates that exposure to many different designs during pre-training enables faster generation of higher quality placements for new unseen blocks.

Figure 3.6 shows the convergence plots for training from scratch vs. training from a pre-trained policy network for Ariane RISC-V CPU [234]. Not only does the pre-trained policy start with a lower placement cost, but it also converges more than 30 hours faster than the policy trained from scratch.

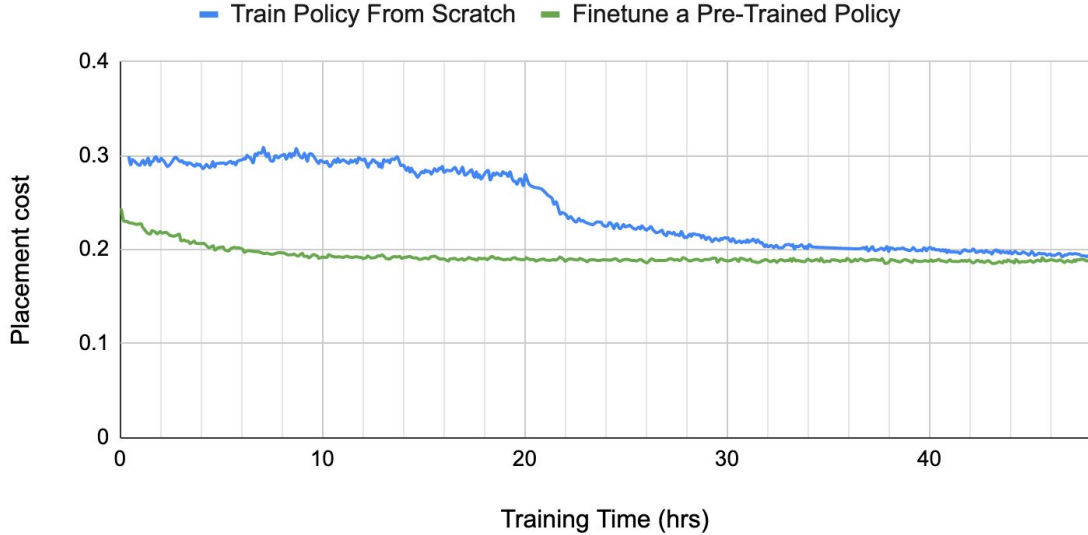


Figure 3.6: Convergence plots on Ariane RISC-V CPU. Here, we plot the placement cost of training a policy network from scratch vs. finetuning a pre-trained policy network for a block of Ariane RISC-V CPU.

3.7.5 Learning from Larger Datasets

In the following, we explore the impact of the training data on the learning ability of our policy. TPU chip blocks are quite diverse, and we carefully selected blocks across a representative range of functionalities (e.g., on-chip and inter-chip network blocks, computation cores, memory controllers, data transport buffers and logic, and various interface controllers), saturations (ratio of total area of macros to that of the canvas, $< 30\%$, $30 - 60\%$, $> 60\%$), and macro counts (up to 107). The small training set contains 2 blocks, the medium set contains 5 blocks, and the large one contains 20 blocks. As we pre-train on more chip blocks, we are able to more quickly generate higher quality placements for new unseen chip blocks. Figure 3.7 shows the impact of a larger training set on performance. As we increase the training set from 2 blocks to 5 blocks and finally to 20 blocks, the policy network generates better placements both at zero-shot and after being finetuned for the same number of hours. This suggests that as we expose the policy network to a greater variety of distinct chip designs, the policy network becomes less prone to overfitting and better at generalizing to new unseen designs.

3.8 Discussion

3.8.1 Use in a Production Setting

Our method was used in the product tapeout of the last four generations of TPU. We fully automated the placement process through PlaceOpt, at which point the design was sent to a third party for post-placement optimization, including detailed routing, clock tree synthesis, and post-clock optimization.

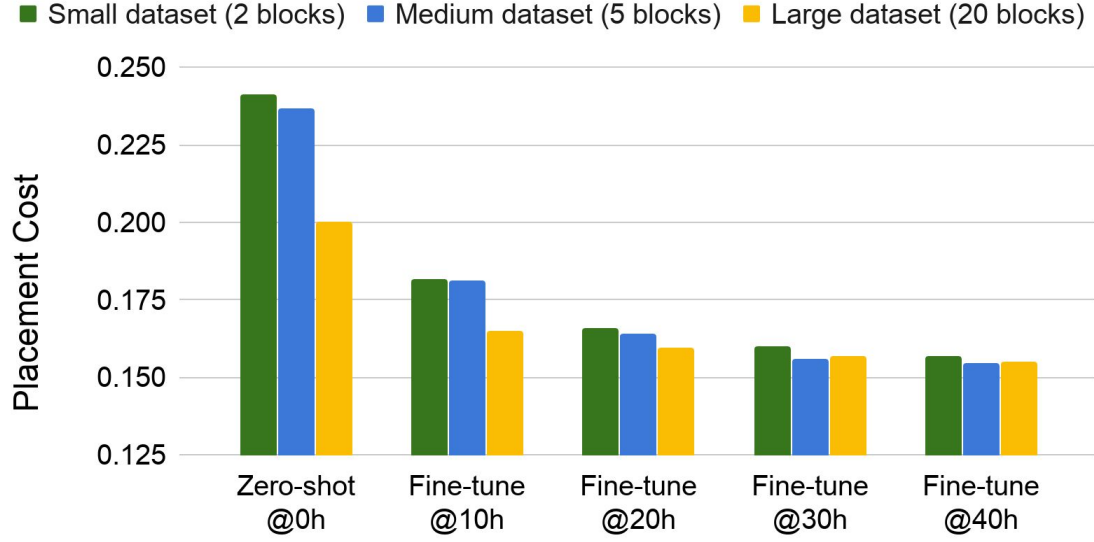


Figure 3.7: Effect of pre-training dataset size. We pre-train the policy network on three different training datasets (the small dataset is a subset of the medium one, and the medium dataset is a subset of the large one). We then finetune this pre-trained policy network on the test block and report cost at various training durations. As the dataset size increases, both the time to convergence and the quality of generated placements increase.

This is a standard practice for many hardware teams, and physical designers spend months iterating with commercial EDA tools to produce designs that meet the strict requirements to move to this next stage.

In the production flow, we use the same reinforcement learning method described in this chapter and the same EDA workflow to place standard cells. Although the RL placements were already comparable to manual designs, we performed an additional fine-tuning step with simulated annealing to further boost performance, which helped to improve macro orientation, as we do not currently perform macro mirroring in RL. Adding this fine-tuning step improved wirelength by an average of 1.07% (stddev=.04%), slightly reduced timing (average 1.18ns reduction in TNS (stddev=2.4ns)), and negligibly affected congestion (less than 0.02% variation in vertical or horizontal congestion in all cases). The resulting end-to-end runtime was 8 hours on average. Since that production launch, we have replaced SA in our production workflow with a greedy postprocessing step that tunes the macro orientation in a few minutes, significantly reducing our end-to-end runtime without degrading quality.

3.8.2 Impact of Cost Trade-offs

In Table 3.5, we perform an ablation study to examine the impact of congestion weight on the quality of post-PlaceOpt results (final quality of result from the commercial EDA tool). As expected,

increasing congestion weight improves both horizontal and vertical congestion up to a point, but results in wirelength degradation, due to the inherent trade-off between these two metrics. A congestion weight of 0.1 represents a sweet spot in this case, as routing congestion is already low, but wirelength has not yet overly degraded, which together contribute to lower TNS and WNS as well.

Congestion Weight	WNS (ps)	TNS (ns)	Wirelength (m)	Congestion Horizontal (%)	Congestion Vertical (%)
0.01	163	22.85	48190736	0.30	0.03
0.1	154	11.80	50841227	0.06	0.03
1.0	118	34.73	53153141	0.07	0.02

Table 3.5: Effect of different cost trade-offs on the post-PlaceOpt performance of Block 1 in Table 3.4. As expected, increasing congestion weight improves both horizontal and vertical congestion up to a point, but results in wirelength degradation, due to the inherent trade-off between these two metrics.

3.8.3 Robustness to Noise

To demonstrate the sensitivity of our method to noise, we performed 8 runs of fine-tuning on Ariane RISC-V block, each time with a different random seed and reported the results (in proxy wirelength, congestion, and density) in Table 3.6. Our evaluations demonstrate that the choice of random seed had negligible impact on all the metrics, including proxy wirelength, congestion, and density, with a standard deviation of .0022 in the overall cost across all runs.

Seed	Proxy Wirelength	Proxy Congestion	Proxy Density
111	0.1187	0.9856	0.5780
222	0.1237	1.0251	0.5691
333	0.1207	0.9456	0.5714
444	0.1189	0.9559	0.5681
555	0.1174	0.9168	0.5561
666	0.1187	0.9676	0.5815
777	0.1200	0.9693	0.5772
888	0.1199	1.0087	0.5819
mean	0.1198	0.9718	0.5729
std	0.0019	0.0346	0.0086

Table 3.6: Sensitivity of results to the choice of random seed as measured on a Ariane RISC-V block. We observed little sensitivity to random seed in any of the three cost functions, though variations in wirelength and density are lower than congestion. The overall cost for these eight runs had a standard deviation of 0.0022.

3.8.4 Generalization vs. Training Data

As we train on more chip blocks, we are able to speed up the fine-tuning process on new blocks and generate higher quality results faster. As discussed earlier, as we increase the training set from 2 blocks (small dataset) to 5 blocks (medium dataset) and finally to 20 blocks (large dataset), the policy network generates better placements both at zero-shot and after being finetuned for the same

number of hours. Figure 3.8 shows the placement cost on one test block (a TPU block not included in training), as the policy network is being pre-trained. We can see that for the small training dataset, the policy network quickly overfits to the training data and performance on the test data degrades, whereas it takes longer for the policy network to overfit on the largest dataset and the policy network pre-trained on this larger dataset yields better results on the test data. This plot suggests that as we expose the policy network to a greater variety of different training blocks, it will take longer for the policy network to pre-train, but the policy network will become less prone to overfitting and better at finding optimized placements for new unseen blocks.

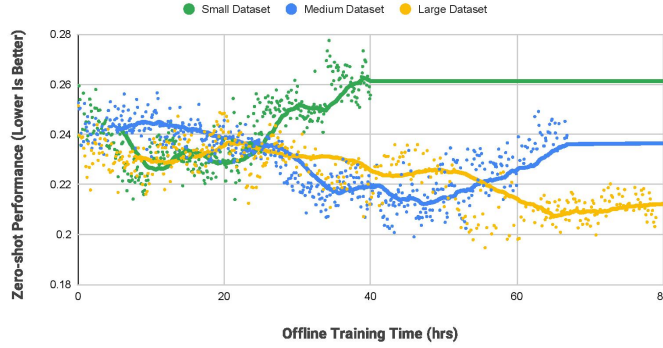


Figure 3.8: Generalization performance as a function of pre-training dataset size. We pre-train the policy network on three different training datasets (the small dataset with 2 blocks is a subset of the medium one with 5 blocks, and the medium dataset is a subset of the large one with 20 blocks). For each policy, at various snapshots during pre-training, we report its inference performance on an unseen test block. As the dataset size increases, both the quality of generated placements on the test block as well as the generalization performance of the policy improves. The policy trained on the largest dataset is most robust to over-fitting.

3.8.5 Insights and Visualizations

Below we share some observations about our method’s behavior that may provide insight into the metrics in Table 3.4.

One observation is that the RL policy tends to place macros on the same datapath close to each other, which results in better timing performance. The Edge-GNN encoder embeds the features of each node by iteratively averaging and applying non-linear transformations to the node’s k -hop neighboring nodes and edges, where k is the number of iterations applied. Therefore, one hypothesis is that the representation of nodes in a given datapath are similar to one another, causing our policy network to generate similar predictions about where they should be placed on the canvas. This naturally results in nodes in the same datapath being placed near to one another, improving timing performance.

Another observation is that our policy learns to reserve sufficient area for the subsequent placement of standard cells, as this effectively optimizes its reward function. Even at zero-shot (meaning that

we run inference on our policy network in less than one second), our method already exhibits this behavior as shown in Figure 3.9.

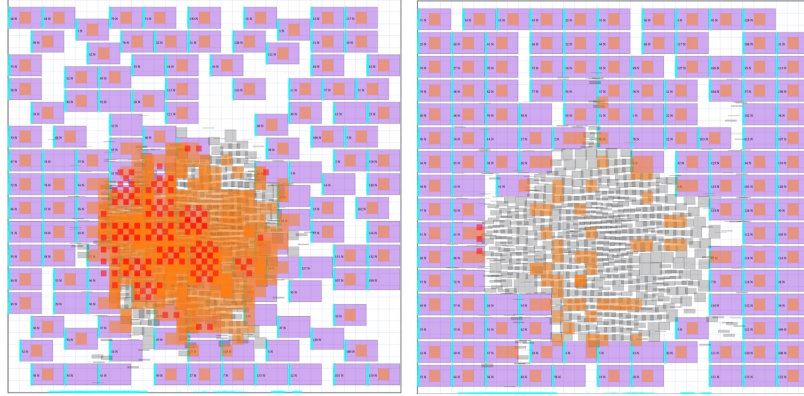


Figure 3.9: Visualization of Ariane placements. On the left, zero-shot placements from the pre-trained policy and on the right, placements from the finetuned policy are shown. The zero-shot placements are generated at inference time on a previously unseen chip. The pre-trained policy network (with no fine-tuning) reserves a convex hull in the center of the canvas in which standard cells can be placed, a behavior which reduces wirelength and which only emerges after many hours of fine-tuning in the policy trained from scratch.

Figure 3.10 juxtaposes a placement generated by a human physical designer (on the left) with that of our method (on the right) for a recent TPU block. The white area shows the macro placements and the green area shows the standard cell placements. Our method creates donut-shaped placements of macros surrounding standard cells, which results in a reduction in the total wirelength. These placement images are blurred to preserve confidentiality.



Figure 3.10: Visualization of a TPU placement. Human expert placements are shown on the left and results from our approach are shown on the right. The white area represents macros and the green area represents standard cells. The figures are intentionally blurred as the designs are proprietary. The wirelength for the human expert design is 57.07m, whereas ours is 55.42m. Furthermore, our method achieves these results in hours, whereas the manual baseline took several weeks.

3.8.6 Implications for a Broader Class of Problems

We believe that the proposed method has broader implications for other stages of chip design and other placement optimization tasks. For example, our zero-shot mode allows design space explorations

through rapid evaluation of computer architectures grounded in the physical reality. Automating and optimizing architectural exploration and its interface with physical design can not only further accelerate the chip design process, but also lead to additional improvements in critical hardware metrics, such as power and timing.

Furthermore, this method is applicable to a broad class of placement optimization problems outside of chip design, such as city planning (e.g., traffic light placement), compiler optimization (e.g., datacenter resource allocation), and environmental engineering (e.g., dam placement).

3.9 Conclusion

In this chapter, we propose an RL-based approach to chip floorplanning that enables domain adaptation. The RL agent becomes better and faster at floorplanning optimization as it places a greater number of chip netlists. We show that our method can generate chip floorplans that are comparable or superior to human experts in under six hours, whereas humans take months to produce acceptable floorplans for modern accelerators. Our method has been used in production to design the next generation of Google TPU.

Appendix

3.A Early Exploration of RL Optimization Algorithms

We felt that it would be valuable to see more side-by-side comparisons of widely used RL algorithms, such as PPO [179], REINFORCE [221], and DQN variants [154, 208, 215] on challenging benchmarks. In the very early days of the AlphaChip project, we ran these comparisons and plotted relative performance. You can see our comparison against DQN variants in the plots below, which show a huge gap between DQN and PPO (0.1642 vs. 0.1112, meaning that PPO achieved 32% higher reward). See Figures 3.A.1 and 3.A.2 below.

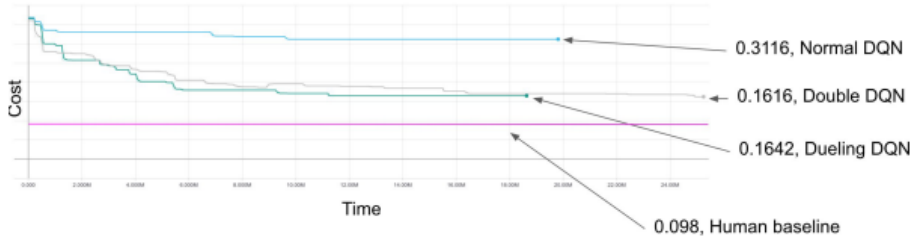


Figure 3.A.1: DQN Variants vs. Human Baseline on an Edge Block. Here, we plot cost (negative reward) for a number of DQN variants, including standard DQN [154], Double DQN [208], and Dueling DQN [215], compared to a human baseline on the task of placement a block of an edge device. Lower cost is better. Double DQN and Dueling DQN achieve similar performance, outperforming vanilla DQN; however, all DQN variants have much higher placement cost than a human expert placement.

Given the strong performance of PPO, we developed a distributed PPO implementation for this task, and have since significantly updated our RL architecture, reward functions, and overall problem formulation.

In recognition of the fact that after such significant changes to our overall method, it is very possible that these other algorithms would now outperform PPO, we did attempt to revive our implementations of other RL optimization algorithms. However, we found that, for example, DQN no longer converges, and we would need to dedicate significant additional development and tuning effort in order to produce fair side-by-side comparisons.

In chip floorplanning, where the search space is massive and the vast majority of solutions are

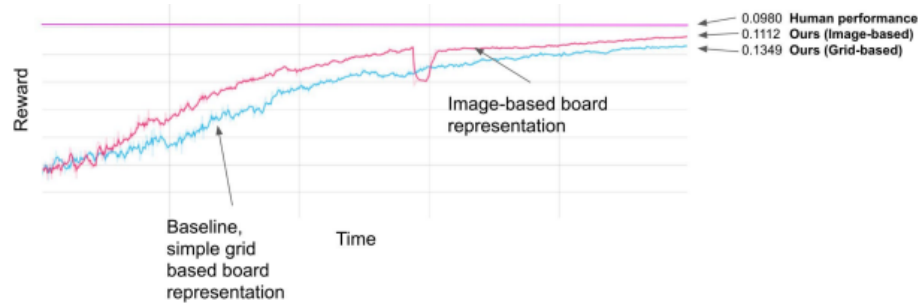


Figure 3.A.2: PPO vs. Human Baseline on an Edge Block. Here, we plot reward for Proximal Policy Optimization (PPO) compared to a human baseline on the task of placement a block of an edge device. Higher reward is better.

invalid, PPO is an effective choice because it allows for safe and efficient exploration, but we do think it is possible that other RL algorithms could enable even better performance.

3.B Exploring RL Convergence Properties

We ran an additional experiment in which we ran the RL method for longer to see the full convergence properties of the pretrained policy, as well as one that was trained from scratch.

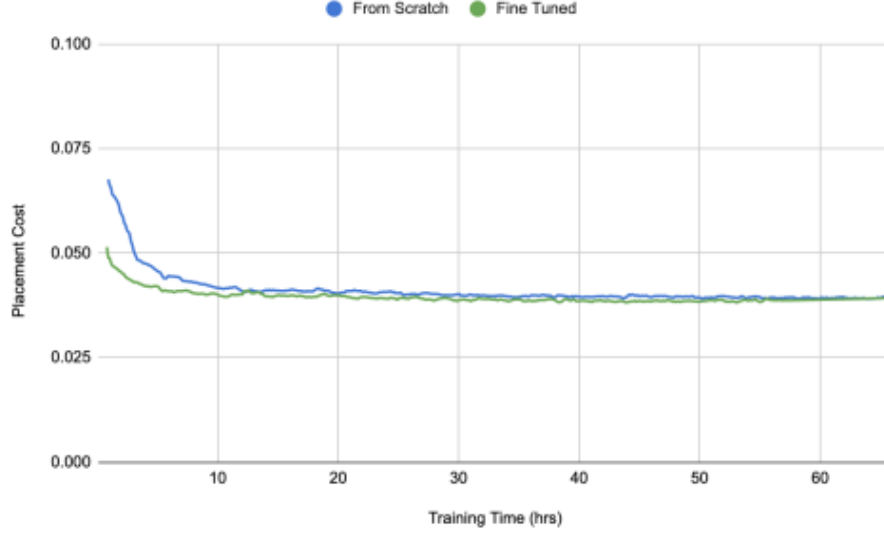


Figure 3.B.1: Placement cost as a function of training times of up to 60 hours, demonstrating that the pre-trained RL agent outperforms the one that is trained from scratch.

Although we weren't able to find the exact checkpoints for the policies in Figure 4 (as those experiments were run nearly five years ago), we did plot placement cost over time for another TPU block. As you can see, both policies have fully converged and produce placements of comparable quality after roughly 60 hours.

It is possible that, on a different block, a policy trained from scratch could eventually overtake the pre-trained policy. However, from a practical perspective, performance gains seen only after multiple days would not be useful to a production chip design team.

3.C Exploring Effect of Input Ordering

Below, we show 10 random orderings for the open-source Ariane RISC-V CPU. As you can see, the standard deviation of wirelength and congestion cost is very low across different macro orderings, perhaps because our edge-based graph convolutional neural network is able to effectively capture netlist topology. This is consistent with what we have observed in earlier work on device placement [150, 147], where using GNNs greatly reduced the impact of placement ordering.

Random Ordering	Proxy Wirelength	Proxy Congestion
#1	0.1144	0.9515
#2	0.1137	0.9378
#3	0.1138	0.9629
#4	0.1131	0.9166
#5	0.1134	0.9150
#6	0.1152	0.9401
#7	0.1127	0.9033
#8	0.1122	0.9294
#9	0.1133	0.9163
#10	0.1145	0.9195
MEAN	0.1136	0.9292
STD	0.0009	0.0186

Table 3.C.1: Ablation Study on the Effect of Macro Ordering. Here, we show the effect of varying the order in which macros are placed by measuring proxy cost given 10 different random macro orderings of the Ariane RISC-V CPU core.

3.D Post-RouteOpt Validation of AlphaChip

After placement optimization, the TPU team hands off the layouts to a third party vendor, which performs detailed routing and sign off.

In Table 3.4, we show results on a previous generation of TPU (TPU-v4), so that we can compare against manual placements generated by the TPU team. On TPU-v4, our method was generating placements with strong QoR metrics (post-PlaceOpt), but their unusual shape was concerning to the TPU team, so they decided to send just one of our placements to this third party vendor for detailed routing and signoff, along with their own manual placement. Our placement met all of the QoR requirements (with slightly better metrics than the human placement) and was closable. Generally speaking, this process is very expensive, but in this one case, both our TPU-v4 placement and the human expert placement were sent to this external vendor for detailed routing and signoff.

In TPU-v5, on the other hand, placements generated by our method were sent for detailed routing and signoff (with no accompanying manual placement), and have been validated all the way to tapeout.

However, to provide insight into the relationship between pre- and post-tool metrics, we provide these metrics for an example TPU-v5 block, as placements generated by our method were taped out in TPU-v5, and therefore were taken all the way post-RouteOpt.

Below, we show a blurred placement of this TPU-v5 block, with the yellow/orange rectangles corresponding to macros. Note that no macros were moved in the course of the external vendor’s detailed routing optimization.



Figure 3.D.1: Image of an AlphaChip layout taped out in TPU-v5. The yellow and orange rectangles correspond to macros. Note that no macros were moved in the course of the external vendor’s detailed routing optimization. The image has been blurred to comply with export control restrictions.

Below, we report the metrics we measured post-PlaceOpt (pre-tool), as well as the post-PlaceOpt (pre-tool) and post-RouteOpt (post-tool) metrics reported by the external vendor. Note that the

vendor did not share wirelength metrics.

Core Block	Step	Wirelength (um)	WNS (ps)	TNS (ns)	Congestion (Vertical)	Congestion (Horizontal)
fp_hde_top	PlaceOpt (Ours)	7.90E+06	-0.002	0	0.00	0.00
fp_hde_top	PlaceOpt (Vendor)	-	-0.008	-0.015	0.00	0.01
fp_hde_top	RouteOpt (Vendor)	-	0.00	0.00	0.00	0.00

Table 3.D.1: Detailed performance metrics for a TPU-v5 block. Here, we report post-PlaceOpt and post-RouteOpt metrics from the external vendor on a block of TPU-v5 (fp_hde_top).

3.E Comparing with Simulated Annealing

In the main body of this chapter, we compared our method against 2 baselines: the academic state-of-the-art RePIace and human expert placements. In this appendix section, we provide an additional comparison with simulated annealing. To generate results for our method, we use the same procedure as in Table 3.4, pre-training a policy on the largest dataset (20 TPU blocks) and then fine-tuning it on the same 5 unseen test blocks.

Simulated Annealing (SA) is known to be a powerful, but slow, optimization method. However, like RL, simulated annealing is capable of optimizing arbitrary non-differentiable cost functions. To show the relative sample efficiency of RL, we ran experiments in which we replaced it with a simulated annealing optimizer. Our SA algorithm works as follows: in each SA iteration (step), we perform $2 \times N$ macro actions (where N is the number of macros). A macro action takes one of three forms: swap, shift, and mirror. Swap selects two macros at random and swaps their locations, if feasible. Shift selects a macro at random and shifts that macro to a neighboring location (left, right, up, or down). Mirror flips a macro at random either across the x-axis, across the y-axis, or across both the x-axis and y-axis. We apply a uniform probability over the three move types, meaning that at each time step there is a $\frac{1}{3}$ chance of swapping, a $\frac{1}{3}$ chance of shifting, and a $\frac{1}{3}$ chance of flipping. After N macro actions, we use a Force-Directed (FD) method to place clusters of standard cells, while keeping macro locations fixed, just as we do in our RL method. For each macro action or FD action, the new state is accepted if it leads to a lower cost. Otherwise, the new state is accepted with a probability of $\exp((prev_{cost} - new_{cost})/t)$, where $t = t_{max} \times \exp(-\log(t_{max}/t_{min}) \times step/steps)$.

To make comparisons fair, we ran 80 SA experiments sweeping different hyperparameters, including max temperature ($\{1 \times 10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 7 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$), max SA episode length ($\{5e4, 1e5\}$), and seed (5 different random seeds), and report the best results in terms of proxy wirelength and congestion costs in Table 3.E.1. Each of the 80 SA workers runs an experiment corresponding to one particular choice of the 5 random seeds, 2 episode lengths, and 8 max temperatures.

	Replacing Deep RL with SA in our framework		Ours	
	Proxy Wirelength	Proxy Congestion	Proxy Wirelength	Proxy Congestion
Block 1	0.048	1.21	0.047	0.87
Block 2	0.045	1.11	0.041	0.93
Block 3	0.044	1.14	0.034	0.96
Block 4	0.030	0.87	0.024	0.78
Block 5	0.045	1.29	0.038	0.88

Table 3.E.1: Performance of our method compared to Simulated Annealing (SA). This table shows proxy wirelength and congestion for each block. Note that because these proxy metrics are relative, comparisons are only valid for different placements of the same block. Even with additional time (18 hours of SA vs. 6 hours of RL), SA generates placements with 14.4% higher wirelength and 24.1% higher congestion on average.

The SA baseline uses more compute (80 SA workers \times 2 CPUs per SA worker \times 18 hours of runtime = 2880 CPU-hours) than our method (16 RL workers \times (1 GPU + 10 CPUs per RL worker) \times 6 hours = 1920 CPU-hours). Here, we are treating the cost of one GPU as roughly 10 times that of a CPU. If we had stopped SA after 12 hours (and if we didn't use early stopping in RL), then the two methods would have used equivalent compute, but the SA results after 12 hours were not even close to competitive. Even with additional time (18 hours of SA vs. 6 hours of RL), SA struggles to produce high-quality placements compared to our approach, generating placements with 14.4% higher wirelength and 24.1% higher congestion on average.

In addition to producing lower quality layouts, we also observed that SA falls far short of RL in its sample efficiency and ability to leverage compute. For example, we plot below the relative sample efficiency of RL vs. SA on a TPU block given different numbers of workers (16, 64, 256, and 1024). SA requires more than 10 \times as many samples (environment calls) to reach the same placement quality when training with 16 workers, and the gap only widens with growing worker counts with SA requiring nearly 100 \times more environment interactions once we scale to 1024 workers.

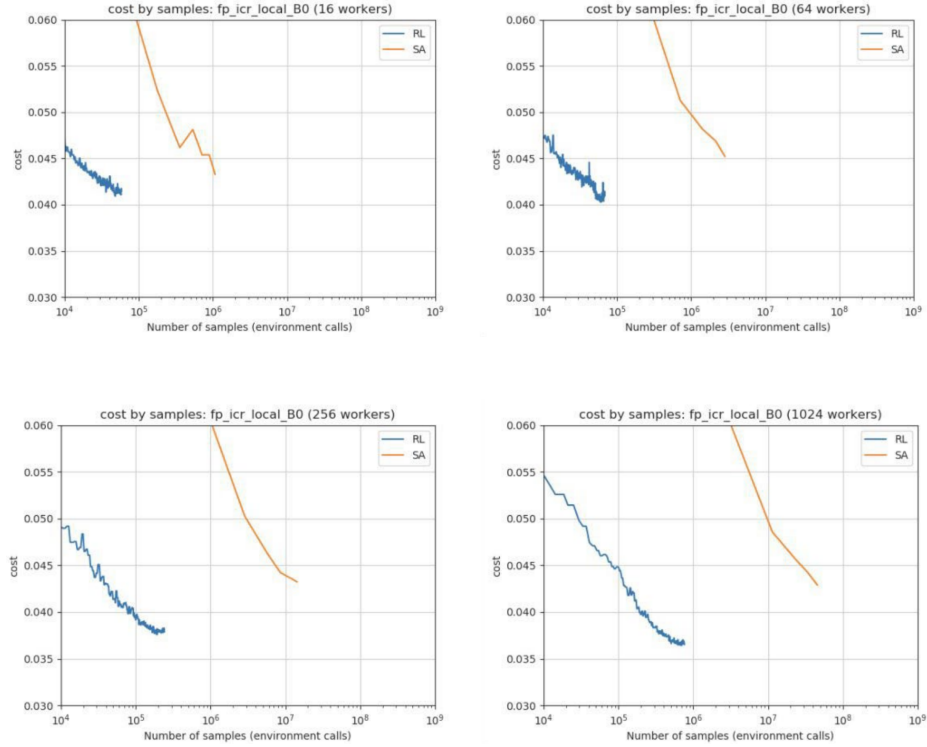


Figure 3.E.1: Sample Efficiency of RL vs. SA. Here, we depict cost as a function of the number of samples (environment calls) for both RL and SA. We show this for a set up with 16 workers, 64 workers, 256 workers, and 1024 workers, in the upper left, upper right, lower left, and lower right, respectively.

Chapter 4

From Scrutiny to Silicon: Post-Publication Discussion and Impact of AlphaChip

4.1 Background

In Chapter 3, we introduced AlphaChip, a deep RL method for chip placement optimization. In this chapter, we describe the post-publication impact and discussion surrounding this work. AlphaChip was one of the first RL methods deployed to solve a real-world engineering problem, and its publication triggered an explosion of work on AI for chip design [229, 227, 233, 228, 42, 66, 23, 195, 29, 47, 65, 216, 28, 144]. Since the work described there was published in *Nature*, AlphaChip has been used in three additional generations of Google’s flagship AI accelerator, the Tensor Processing Unit (TPU). These chips have been manufactured and deployed in data centers all over the world. As shown in Figure 4.1.1, the gap between the performance of AlphaChip and human experts has grown with each successive generation of TPU, going from 10 RL-placed blocks and 3.2% wirelength reduction vs. human experts in TPU v5e, to 15 blocks with 4.5% reduction in TPU v5p, to 25 blocks with 6.2% reduction in Trillium. AlphaChip has also generated superhuman chip layouts for blocks used in datacenter CPUs (Axion) and other unannounced chips across Alphabet. Other organizations have adopted our approach and built on it. For example, MediaTek, a leading global chipmaker, extended AlphaChip to accelerate development of their most advanced chips, while improving power, performance, and area.

Nevertheless, as described in Sutton’s “The Bitter Lesson” [192], there is often reluctance to accept the application of machine learning to new areas, and ultimately this has led to some confusion

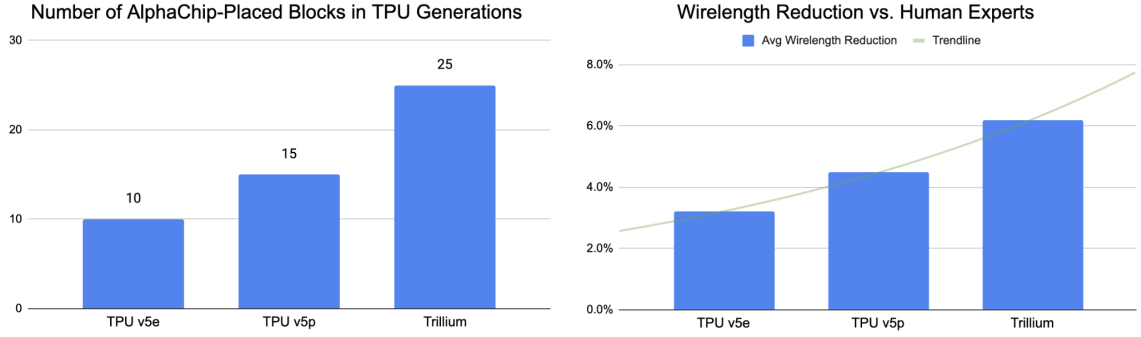


Figure 4.1.1: AlphaChip has been deployed in three additional generations of TPU. In each generation, it has been adopted in a greater proportion of blocks and has outperformed human experts by a wider margin.

around our work, which we summarize below.

These concerns were collected into an article published in the November 2024 issue of *Communications of the ACM* [141], which was presented as a “meta-analysis” of our *Nature* paper and two non-peer-reviewed papers:

- **Cheng *et al.*:** The first is an invited ISPD paper¹ by Cheng *et al.* [39]. This paper did not follow standard machine learning practices, and its reinforcement learning methodology and experimental setup diverged significantly from those described in our *Nature* paper. Nevertheless, its hamstrung version of our method still outperformed RePlace² [48], which was the state of the art when we published in *Nature*.
- **Markov *et al.*:** The second “meta-analyzed” paper is an unpublished PDF with no author list [15], which is described as a “separate evaluation” performed by “Google Team 2”, but was in fact co-authored by the author of the CACM article, Igor Markov³, though this is not disclosed⁴. This paper did not meet Google’s bar for publication. In 2022, it was reviewed by an independent committee at Google, which determined that “the claims and conclusions in the draft are not scientifically backed by the experiments” [163] and “as the [AlphaChip] results on their original datasets were independently reproduced, this brought the [Markov *et al.*] RL results into question” [163]. We provided the committee with one-line scripts that generated significantly better RL results than those reported in Markov *et al.*, outperforming their “stronger” simulated annealing baseline. We still do not know how Markov and his collaborators produced the numbers in their paper.

¹Invited papers at ISPD are not peer-reviewed.

²Incidentally, RePlace, as noted in a footnote of Cheng *et al.*, is unable to produce any result at all for 2 out of the 6 test cases in its main data table.

³Markov did not disclose anywhere in his “meta-analysis” that he is an author of one of the two “separate evaluations”. He also omitted his name from the paper’s authors in the references section, and linked only to an anonymous PDF. When questioned on LinkedIn, Markov admitted his authorship, but later deleted the post.

⁴Markov also failed to disclose his role as a high-level employee at Synopsys, a company which licenses commercial tools that compete with our open-source method.

Markov’s “meta-analysis” offers one additional source of concern regarding our paper: a “whistle-blower” within Google. However, this “whistleblower” admitted to a Google investigator that he had no reason to believe fraud occurred: “he stated that he suspected that the research being conducted by Goldie and Mirhoseini was fraudulent, but also stated that he did not have evidence to support his suspicion of fraud” [123].

In his “meta-analysis”, Markov speculates wildly and without evidence about fraud and scientific misconduct, none of which occurred. Most of Markov’s criticisms are of this form: it does not look to him like our method *should* work, and therefore it *must not* work, and any evidence suggesting otherwise is fraud. *Nature* investigated Markov’s concerns, found them to be entirely without merit, and published an Addendum upholding our work at the conclusion of this process [75].

As an example, in the opening paragraph of his conclusions, Markov states (emphasis his): “In the paper, we find a smorgasbord of *questionable practices in ML* [26]⁵ including irreproducible research practices, multiple variants of cherry-picking, misreporting, and likely data contamination (leakage).” We did not engage in any of these practices, or any other form of scientific misconduct, and Markov provides no evidence for these allegations. Nowhere in Markov’s paper does he describe any form of alleged cherry-picking, let alone multiple variants, nor does he provide evidence. Nor does he describe any form of alleged “misreporting,” or explain what he means by this, or provide evidence. Nor does he provide any evidence of data contamination (leakage), aside from his speculation that it would have improved our results if it had occurred. Many of these allegations appear *for the first time* in his “Conclusions” section!

In an effort to discredit our TPU deployments, Markov also suggests that Google must just be “dogfooding” our method, allowing inferior AlphaChip placements to be used in TPU in order to prop up our research paper. This is untrue, and absurd on its face. Google cares far more about the efficiency of TPU designs – a multi-billion-dollar project that is central to Google’s cloud and AI initiatives – than it does about a research paper⁶.

For clarity, we present a timeline of events, including non-confidential deployments⁷:

- **Apr 2020:** Released arXiv preprint of our *Nature* paper [16].
- **Aug 2020:** 10 AlphaChip layouts taped out in TPU v5e.
- **Jun 2021:** Published *Nature* article [149].
- **Sep 2021:** 15 AlphaChip layouts taped out in TPU v5p.

⁵Note that Markov’s citation 26 has nothing to do with our paper, though readers may mistakenly believe that it offers corroboration.

⁶In reality, we had to work for a long time to build enough trust for the TPU team to use our layouts, even after AlphaChip was outperforming human experts on the metrics, and this makes sense – their job is to deliver TPU chips and make them as efficient and reliable as possible, and they cannot afford to take unnecessary risks.

⁷AlphaChip has been deployed in other hardware across Alphabet that we cannot yet disclose.

- **Jan 2022 - Jul 2022:** Open-sourced AlphaChip [79], after ensuring compliance with export control restrictions and excising internal dependencies. This involved independent replication of the results in our *Nature* paper by another team at Google. See Section 4.3.
- **Feb 2022:** Independent committee within Google declined to publish Markov *et al.* as the data did not support its claims and conclusions [163].
- **Oct 2022:** 25 AlphaChip layouts taped out in Trillium.
- **Feb 2023:** Cheng *et al.* posted on arXiv [39], claiming to perform “massive reimplementa-
tion” of our method, despite it being fully open-source. As discussed in Sections 4.2 and 4.A, Cheng *et al.* did not run our method as described in *Nature*, among other issues.
- **Jun 2023:** Markov released arXiv preprint of his “meta-analysis” [140].
- **Sep 2023:** *Nature* posted Editor’s note stating that they are investigating our paper, and initiated second peer review process [149].
- **Mar 2024:** 7 AlphaChip layouts adopted in Google Axion Processors (ARM-based CPU).
- **Apr 2024:** *Nature* completed its investigation and post-publication review, and found entirely in our favor, concluding that “the best way forward is to publish an update to the paper in the form of an Addendum (not a ‘Correction’, as we have established that there is little that actually needs correcting).” [241]
- **Sep 2024:** *Nature* published Addendum upholding our work [75], removed Editor’s note.
- **Sep 2024:** SVP of MediaTek announced that they extended AlphaChip to accelerate develop-
ment of their most advanced chips [74].
- **Nov 2024:** Markov republished his “meta-analysis”, though his concerns were already found to be without merit during *Nature*’s investigation and second peer review process.
- **April 2025:** AlphaChip layouts used in Ironwood (latest public TPU as of May 2025).

In brief, Markov’s CACM paper contains no original data, and is a “meta-analysis” of just two papers. The first is presented with no author list (though Markov was an author), was never published, made claims that were not scientifically backed by the data, and could not be reproduced. The second, Cheng *et al.*, is the only substantive content in Markov’s “meta-analysis”, so we devote the remainder of this chapter to describing significant issues in its purported reproduction of our method.

4.2 Errors in Attempted Reproduction of Our Method

Cheng *et al.* claim to evaluate our method against alternative approaches on new test cases. Unfortunately, Cheng *et al.* did not run our method as described in *Nature*, so it is unsurprising that they report different results. In this section, we describe major errors in their purported reproduction:

- **Did not pre-train the RL method.** The ability to learn from prior experience is the key advantage of our learning-based method, and to remove it is to evaluate a different and inferior approach. Incidentally, pre-training also gives rise to the impressive capabilities of large language models like Gemini [199] and ChatGPT [161] (the “P” in “GPT” stands for “pre-trained”). See Section 4.2.1.
- **Used an order of magnitude fewer compute resources:** 20x fewer RL experience collectors (26 vs 512 in *Nature*) and 2x fewer GPUs (8 vs 16 in *Nature*). See Section 4.2.2.
- **Did not train to convergence.** Training to convergence is standard practice in machine learning, as not doing so is well known to harm performance [4]. See Section 4.2.3.
- **Evaluated on non-representative, irreproducible benchmarks.** Cheng *et al.*’s benchmarks have much older and larger technology node sizes (45nm and 12nm vs sub-7nm in *Nature*), and differ substantially from a physical design perspective. Additionally, the authors were unable or unwilling to share the synthesized netlists necessary to replicate the results in their main data table. See Sections 4.2.4 and 4.3.2.
- **Performed “massive reimplementation” of our method,** which may have introduced errors. We recommend instead using our open-source code. See Section 4.3.

These major methodological differences unfortunately invalidate Cheng *et al.*’s comparisons with and conclusions about our method. If Cheng *et al.* had reached out to the corresponding authors of the *Nature* paper⁸, we would have gladly helped them to correct these issues prior to publication⁹.

4.2.1 No Pre-Training Performed for RL Method

Unlike prior approaches, AlphaChip is a learning-based method, meaning that it becomes better and faster as it solves more instances of the chip placement problem. This is achieved by pre-training, which consists of training on “practice” blocks (training data) prior to running on the held-out test cases (test data).

⁸Prior to publication of Cheng *et al.*, our last correspondence with any of its authors was in August of 2022 when we reached out to share our new contact information.

⁹In contrast, prior to publishing in *Nature*, we corresponded extensively with Andrew Kahng, senior author of Cheng *et al.* and of the prior state of the art (RePIAce), to ensure that we were using the appropriate settings for RePIAce.

As we showed in Figure 5 of our *Nature* paper (reproduced below as Figure 4.2.1), the larger the training dataset is, the better the method becomes at placing new blocks. As described in our *Nature* article, we pre-trained on 20 blocks in our main data table (*Nature* Table 1).

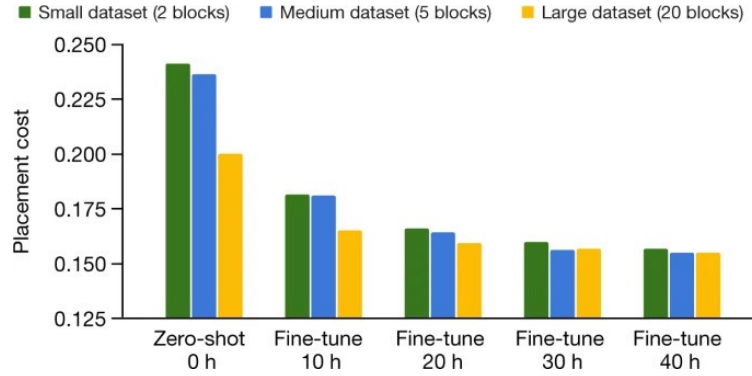


Figure 4.2.1: Figure 5 of the *Nature* paper (reproduced above) shows performance gains from pre-training on a larger number of blocks. As we scale up the pre-training dataset size, the RL agent’s performance improves.

Cheng *et al.* did not pre-train at all (i.e., no training data), meaning that the RL agent had never seen a chip before and had to learn how to perform placement from scratch for each of the test cases. This **removed the key advantage of our method**, namely its ability to learn from prior experience.

By analogy to other well-known work on reinforcement learning, this would be like evaluating a version of AlphaGo [186] that had never seen a game of Go before (instead of being pre-trained on millions of games), and then concluding that AlphaGo is not very good at Go.

We discussed the importance of pre-training at length in our *Nature* paper (e.g., the word “pre-train” appeared 37 times), and empirically demonstrated its impact. For example, *Nature* Figure 4 (reproduced here as Figure 4.2.2) showed that pre-training improves placement quality and convergence speed. On the open-source Ariane RISC-V CPU [62], it took 48 hours for the non-pretrained RL policy to approach what the pre-trained model could produce in 6 hours. As described in our *Nature* paper, we pre-trained for 48 hours for the results in our main data table, whereas Cheng *et al.* pre-trained for 0 hours.

Our open-source repository [79] enables full reproduction of the methods described in our *Nature* paper (see Section 4.3). Cheng *et al.* have attempted to excuse their lack of pre-training by suggesting that our open-source repository does not support pre-training [39], but this is incorrect. Pre-training is simply running the method on multiple examples, and this has been always been supported.

4.2.2 RL Method Provided with Far Fewer Compute Resources

In Cheng *et al.*, the RL method is provided with 20x fewer RL experience collectors (26 vs 512 in *Nature*) and half as many GPUs (8 vs 16 in *Nature*). Using less compute is likely to harm performance,

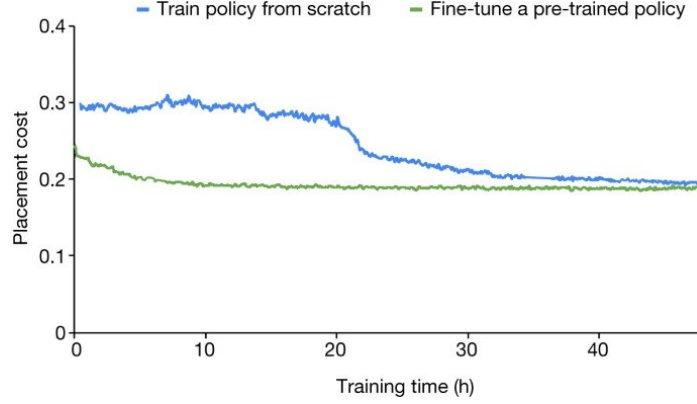


Figure 4.2.2: Figure 4 of the *Nature* paper (reproduced above) showed that pre-training improves convergence speed compared to starting from a randomly initialized policy. On the open-source Ariane RISC-V CPU, the randomly initialized policy took 48 hours to approach what the pre-trained policy could produce in 6 hours.

or require running for considerably longer to achieve the same (or worse) performance.

As shown in Figure 4.2.3 (reproduced from a follow-up paper [232]), training on a larger number of GPUs speeds convergence and yields better final quality. If Cheng *et al.* had matched the experimental settings described in *Nature*, this would likely have improved their results.

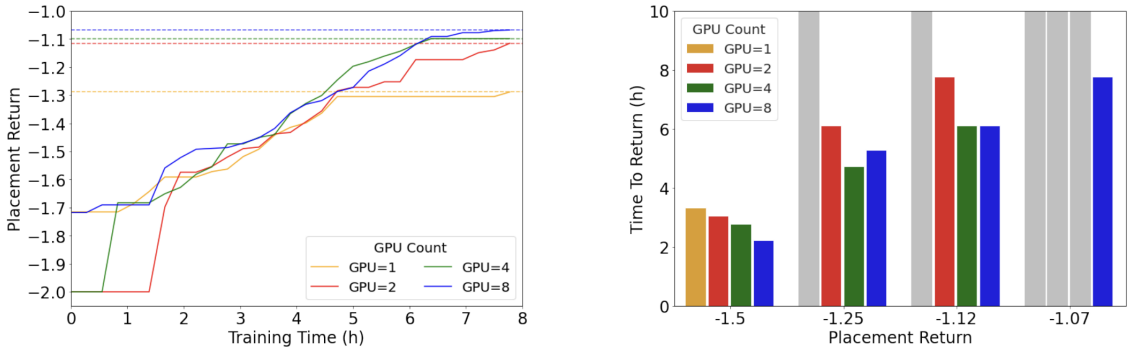


Figure 4.2.3: Figure 6 from a follow-up paper [232] (reproduced above) demonstrated that speed and quality improve with additional compute resources. Left: Placement return (higher is better) vs. training time as a function of the number of GPUs. An infeasible placement receives a -2 placement return. Increasing the number of GPUs results in better final placements. Right: Time to reach a given placement return as a function of the number of GPUs. The grey bars indicate that the experiment did not reach a specific return value. The best placement return -1.07 can only be achieved with GPU=8, the largest setting in this experiment.

4.2.3 RL Method Not Trained to Convergence

As a machine learning model trains, loss typically decreases and then asymptotes, representing “convergence” — the model has learned what it can about the task it is performing. Training to convergence is standard practice in machine learning, and not doing so is well known to harm

performance [4].

Cheng *et al.* did not train to convergence on any of the four blocks for which convergence plots were provided on their accompanying project site [159] (no plots were provided for BlackParrot-NG45 or Ariane-NG45).

Figure 4.2.4 shows the convergence plots from Cheng *et al.*’s project site, and Table 4.2.1 summarizes the information available. For all four blocks with convergence plots (Ariane-GF12, MemPool-NG45, BlackParrot-GF12, and MemPool-GF12), training was cut off at a relatively low step count (350k, 250k, 160k, and 250k steps, respectively)¹⁰. Following standard machine learning practices would likely improve performance on these test cases.

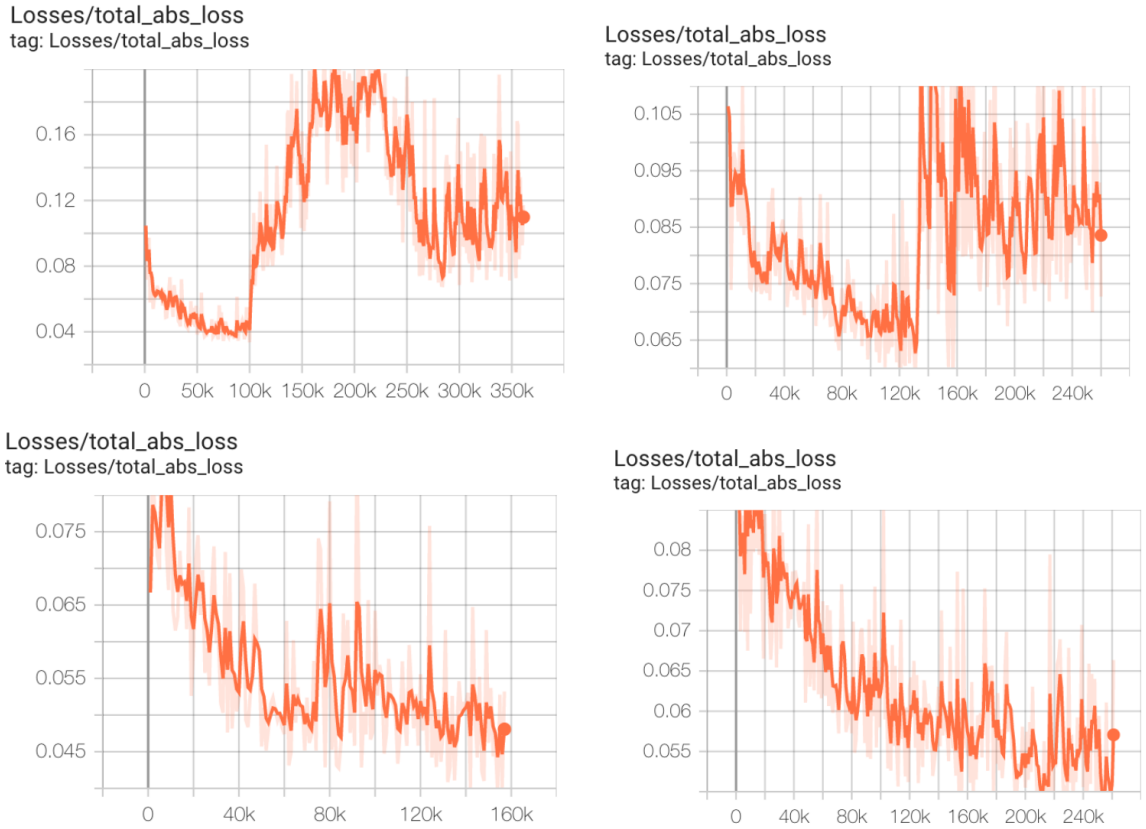


Figure 4.2.4: Convergence plots from Cheng *et al.*’s project site. On Ariane-NG45 (top left) and MemPool-NG45 (top right), there is an odd divergence at around 100k steps, but loss appears to be trending downwards and would likely have improved with further training. On BlackParrot-GF12 (bottom left) and MemPool-GF12 (bottom right), the model has not yet converged and would likely benefit from additional training time as well.

¹⁰Although Cheng *et al.*’s Figure 4 appears to show convergence on Ariane-NG45 after 1M steps, it omits most components of the total training loss, depicting only wirelength, density, and congestion costs. However, total loss is composed of entropy regularization loss, KL penalty loss, L2 regularization loss, policy gradient loss, and value estimate loss. See open-source code for details of training loss: https://github.com/google-research/circuit_training/blob/90f8e0e939c3038e43db63d2cf1ff570e525547a/circuit_training/learning/agent.py#L408. Cheng *et al.* did not provide the Tensorboard for this block, and as shown in Table 4.2.1, all other blocks were run for far fewer than 1M steps.

Block Name from Cheng <i>et al.</i> 's Table 1	Tensorboard?	Num Steps	Total Loss Curve
Ariane-NG45	No	1M	No Tensorboard.
BlackParrot-NG45	No	?	No Tensorboard.
MemPool-NG45	Yes	250k	Divergence at 100k steps, clearly has not converged.
Ariane-GF12	Yes	350k	Divergence at 130k steps, clearly has not converged.
BlackParrot-GF12	Yes	160k	Still converging, training stopped prematurely.
MemPool-GF12	Yes	250k	Still converging, training stopped prematurely.

Table 4.2.1: Cheng *et al.* did not train properly on any of the test cases for which Tensorboards were provided on the accompanying project site.

4.2.4 Test Cases Not Representative of Modern Chips

In our *Nature* paper, we report results on Tensor Processing Unit (TPU) blocks with sub-7nm technology node size, which is typical of modern chips. In contrast, Cheng *et al.* reports results on older technology node sizes (45nm and 12nm), which differ substantially from a physical design perspective; for example, at sub-10nm, multiple patterning is typically used [52, 220], causing routing congestion issues to emerge at lower density. Therefore, for older technology node sizes, our method may benefit from adjustment to the congestion or density components of its reward function¹¹. We have not focused on applying our technique to designs with older nodes because all of our work is at 7nm, 5nm, and more recent processes, though we would welcome contributions from the community on this front.

4.3 Transparency & Reproducibility

4.3.1 AlphaChip is Fully Open-Source

We have open-sourced a software repository [79] to fully reproduce the methods described in our *Nature* paper. Every line of our RL method is freely available for inspection, execution, or modification, and we provide source code or binaries to perform all preprocessing and postprocessing steps. Open-sourcing the code took over a year of effort by the TF-Agents team¹², and included independent replication of our methodology and the results in our *Nature* paper. From our open-source repository:

“Open-sourcing our code involved partnering with another team at Google (TF-Agents).

¹¹Google engineers suggested this, but their guidance was not followed (see Section 4.A.4).

¹²TensorFlow Agents is a reinforcement learning infrastructure team at Google, which provides open-source libraries. See <https://www.tensorflow.org/agents>.

TF-Agents first replicated the results in our Nature article using our codebase, then reimplemented our method and replicated our results using their own implementation, and then open-sourced their implementation as it does not rely on any internal infrastructure.”

Cheng *et al.* unnecessarily “reverse-engineered” two functions that we provide as binaries for performance optimization (the proxy cost function and force-directed (FD) standard cell placer). As discussed in an MLCAD 2021 paper [93], we now recommend using DREAMPlace [132] for standard cell placement, rather than FD, as it yields superior performance. We provide the legacy FD binary for the sole purpose of enabling exact reproduction of our method as published in *Nature*.

Regarding public benchmarks, we reported results on the open-source Ariane RISC-V CPU [62] in *Nature*. Additionally, in a follow-up paper at MLCAD 2021 [93], we evaluated on the open-source ISPD 2015 contest benchmark [27]. Because we have open-sourced our code, the community is free to follow our methodology and evaluate our method on any public benchmark.

4.3.2 Claims They Cannot Share Their “Open” Test Cases

One of the criticisms put forth in Cheng *et al.* was that the *Nature* evaluation was done on proprietary TPU blocks (in addition to the open-source Ariane block that was also evaluated, and the public ISPD 2015 benchmark in a follow-up publication [93]). Cheng *et al.* claimed to evaluate on a set of open test cases to improve reproducibility, but when we corresponded with the authors, they were unable or unwilling to provide the synthesized netlists necessary to replicate their results on the “open” test cases in their main data table (Table 1).

Unfortunately, this means that we cannot replicate any of the results in Cheng *et al.*’s Table 1:

- **GF12 (12nm)**: These test cases are proprietary and unavailable to the public, and Cheng *et al.*’s results are obfuscated, meaning that even if an external researcher were to obtain access, a direct comparison would still be impossible.
- **NG45 (45nm)**: Cheng *et al.* have not shared the synthesized netlists necessary to reproduce their NG45 results, despite 10+ requests since February 2024. Note that other papers evaluate on the NG45 blocks, but their results are inconsistent with those in Cheng *et al.*’s Table 1 (e.g., see Table 2 of AutoDMP [5]), underscoring reproducibility challenges.

Notably, other researchers have also repeatedly requested access to these test cases and to the output placements referenced in the paper, but Cheng *et al.* have so far been unable to release them¹³, meaning that no one is able to reproduce their findings on these “open” testcases.

It is unfortunate that modern chip IP is sensitive and proprietary, and to our knowledge, there are no open benchmarks available for cutting edge processes. We encourage the chip design community

¹³For example, see <https://github.com/TILOS-AI-Institute/MacroPlacement/issues/78> and <https://github.com/TILOS-AI-Institute/MacroPlacement/issues/67>.

to create more open designs for modern sub-7nm processes, as this will help push the field forward. At the moment, fully open designs are typically 28nm, 45nm, or even 130nm, and many physical design issues are quite different than in sub-7nm processes.

4.4 Discussion

In Cheng *et al.*'s attempt to reassess our work, the authors did not run our method as described in *Nature* (e.g., they performed no pre-training, used substantially less compute, and did not train to convergence), reported results on benchmarks that are neither representative nor reproducible, and ran questionable ablation/correlation studies.

In his paper [141], Markov published baseless allegations of fraud based on a “meta-analysis” of Cheng *et al.* (which did not reproduce our methodology) and an anonymous PDF (that Markov actually coauthored), whose results could not be reproduced and for which “the claims and conclusions in the draft are not scientifically backed by the experiments” [163].

Meanwhile, AlphaChip has inspired an explosion of work on AI for chip design, and its superhuman layouts have been taped out in four generations of TPU deployed in Google datacenters all over the world, as well as other chips across Alphabet and by external chipmakers. We look forward to seeing AI continue to transform all aspects of hardware design, just as advances in hardware have revolutionized AI.

4.5 Moving Forward: AlphaChip's Broader Impact

AlphaChip's impact can be seen through its applications across Alphabet, the research community and the chip design industry. Beyond designing specialized AI accelerators like TPUs, AlphaChip has generated layouts for other chips across Alphabet, such as Google Axion Processors, Google's first Arm-based general-purpose data center CPUs.

External organizations are also adopting and building on AlphaChip. For example, MediaTek, one of the top chip design companies in the world, extended AlphaChip to accelerate development of their most advanced chips while improving power, performance and chip area.

AlphaChip has triggered an explosion of work on AI for chip design, and has been extended to other critical stages of chip design, such as logic synthesis [44] and macro selection [124].

4.6 External Perspectives from Academic and Industry Leaders

Below we share perspectives from industry and academic leaders on the research and production impact of AlphaChip [74]:

“AlphaChip’s groundbreaking AI approach revolutionizes a key phase of chip design. At MediaTek, we’ve been pioneering chip design’s floorplanning and macro placement by extending this technique in combination with the industry’s best practices. This paradigm shift not only enhances design efficiency, but also sets new benchmarks for effectiveness, propelling the industry towards future breakthroughs.”

— **SR Tsai, Senior Vice President of MediaTek**

“AlphaChip has inspired an entirely new line of research on reinforcement learning for chip design, cutting across the design flow from logic synthesis to floor planning, timing optimization and beyond. While the details vary, key ideas in the paper including pretrained agents that help guide online search and graph network based circuit representations continue to influence the field, including my own work on RL for logic synthesis. If not already, this work is poised to be one of the landmark papers in machine learning for hardware design.”

— **Siddharth Garg, Professor of Electrical and Computer Engineering, NYU**

“AlphaChip demonstrates the remarkable transformative potential of Reinforcement Learning (RL) in tackling one of the most complex hardware optimization challenges: chip floorplanning. This research not only extends the application of RL beyond its established success in game-playing scenarios to practical, high-impact industrial challenges, but also establishes a robust baseline environment for benchmarking future advancements at the intersection of AI and full-stack chip design. The work’s long-term implications are far-reaching, illustrating how hard engineering tasks can be reframed as new avenues for AI-driven optimization in semiconductor technology.”

— **Vijay Janapa Reddi, John L. Loeb Associate Professor of Engineering and Applied Sciences, Harvard University**

“Reinforcement learning has profoundly influenced electronic design automation (EDA), particularly by addressing the challenge of data scarcity in AI-driven methods. Despite obstacles including delayed rewards and limited generalization, research has proven reinforcement learning’s capability in complex electronic design automation tasks such as floorplanning. This seminal paper has become a cornerstone in reinforcement learning-electronic design automation research and is frequently cited, including in my own work that received the Best Paper Award at the 2023 ACM Design Automation Conference.”

— **Professor Sung-Kyu Lim, Georgia Institute of Technology**

“There are two major forces that are playing a pivotal role in the modern era: semiconductor chip design and AI. This research charted a new path and demonstrated ideas that enabled the electronic design automation (EDA) community to see the power of AI and reinforcement learning for IC design. It has had a seminal impact in the field of AI for chip design and has been critical in influencing our thinking and efforts around establishing a major research conference like IEEE LLM-Aided Design (LAD) for discussion of such impactful ideas.”

— **Ruchir Puri, Chief Scientist, IBM Research; IBM Fellow**

4.7 Conclusion

We believe AlphaChip has the potential to optimize every stage of the chip design cycle, from computer architecture to manufacturing — and to transform chip design for custom hardware found in everyday devices such as smartphones, medical equipment, agricultural sensors and more.

Future versions of AlphaChip are now in development and we look forward to working with the community to continue revolutionizing this area and to bring about a future in which chips are even faster, cheaper and more power-efficient.

Appendix

4.A Other Discussions

In this appendix, we describe other concerns with Cheng *et al.*, including its comparison with closed-source commercial autoplacers, its contrived “ablation” of initial placement in standard cell cluster rebalancing, its flawed correlation study, and its erroneous claim of validation by Google engineers.

4.A.1 Inappropriate Comparison With Commercial Autoplacers

Cheng *et al.* compares a severely weakened RL method against unpublished, closed-source, proprietary software released years after our method was published. This is not a reasonable way to evaluate our method – for all we know, the closed-source tool could have built directly on our work.

In May of 2020, we performed a blind internal study¹⁴ comparing our method against the latest version of two leading commercial autoplacers. Our method outperformed both, beating one 13 to 4 (with 3 ties) and the other 15 to 1 (with 4 ties). Unfortunately, standard licensing agreements with commercial vendors prohibit public comparison with their offerings.

4.A.2 Contrived “Ablation” of Initial Placement in Standard Cell Cluster Rebalancing

Prior to running the methods evaluated in our *Nature* paper, an approximate initial placement from physical synthesis, the previous step of the chip design process [196], was used to resolve imbalances in the sizes of standard cell clusters from hMETIS [108].

Cheng *et al.* ran an “ablation” study on a single block (Ariane-NG45). Instead simply skipping the cluster rebalancing step, they tried placing all chip components on top of each other in the

¹⁴Our blind study compared RL to human experts and commercial autoplacers on 20 TPU blocks. First, the physical design engineer responsible for placing a given block ranked anonymized placements from each of the competing methods, evaluating purely on final QoR metrics with no knowledge of which method was used to generate each placement. Next, a panel of seven physical design experts reviewed each of the rankings and ties. The comparisons were unblinded only after completing both rounds of evaluation. The result was that the best placement was produced most often by RL, followed by human experts, followed by commercial autoplacers.

lower-left corner¹⁵, causing the rebalancing step to produce degenerate standard cell clusters. When this harmed performance, Cheng *et al.* concluded that our RL agent was somehow making use of initial placement information, even though it does not have access to the initial placement and does not place standard cells.

We ran an ablation study which eliminated any use of initial placement whatsoever, and observed no degradation in performance (see Table 4.A.1). We simply skipped the cluster rebalancing step and instead reduced hMETIS’s cluster “unbalancedness” parameter to its lowest setting (UBFactor=1)¹⁶, which causes hMETIS to generate more balanced clusters [108]. This ancillary preprocessing step has been documented and open-sourced since June 10, 2022 [79], but is unnecessary and has already been removed from our production workflow.

TPU-v6 Block	wirelength	wns	tns	density	congestion (H)	congestion (V)
Clustering with initial placement	5,176	-0.046	-2.466	23.830	0.01	0.01
Clustering with no initial placement	5,133	-0.048	-2.583	23.827	0.01	0.01

Table 4.A.1: RL results after clustering standard cells with and without the initial placement. Lower magnitude is better for all metrics. Clustering without initial placement does not appear to harm performance.

4.A.3 Flawed Study of Correlation Between Proxy Cost and Final Metrics

Cheng *et al.* claimed that our proxy costs are not well-correlated with final metrics, but their correlation study actually showed a weak but positive correlation between overall proxy cost and all final metrics except standard cell area (see Cheng *et al.*’s Table 2, reproduced here as Figure 4.A.1). Note that we treat area as a hard constraint, and therefore do not optimize for it.

Proxy Cost Element	Std Cell Area	rWL	Power	WNS	TNS
Wirelength	-0.221	0.317	-0.144	0.163	0.317
Congestion	-0.029	0.086	-0.010	0.105	-0.048
Density	0.096	0.230	0.096	0.268	0.077
Proxy	-0.010	0.257	0.048	0.200	0.048

Figure 4.A.1: Cheng *et al.*’s Table 2 (reproduced above, emphasis ours) showed a weak but positive correlation between overall proxy cost and final metrics, except std cell area, which we treat as a hard constraint and do not optimize.

Proxy costs used in ML-based optimization are often only weakly correlated with the target

¹⁵Cheng *et al.* also tried placing all components on top of each other in the upper-right corner and on a single point in the center of the canvas. Unsurprisingly, this yielded the same degenerate results.

¹⁶UBfactor is a parameter that ranges from 1 to 49, where lower settings instruct hMETIS to prioritize balanced cluster sizes. UBfactor was set to 5 in our *Nature* paper.

objective. For example, large language models like Gemini [199] and ChatGPT [161] are trained to guess the next word in a sequence, which is an intrinsically noisy signal.

Additionally, Cheng *et al.*'s correlation study made some surprising choices:

- Cheng *et al.* only report correlation for proxy costs below 0.9 and provide no justification for this decision. This threshold excludes the majority of their own results (e.g., see Cheng *et al.*'s Table 1).
- The correlation study considers only a single 45nm test case (Ariane-NG45). NG45 is a much older technology node size and the congestion and density components of the overall cost function should probably be adjusted for better correlation (see Section 4.2.4).

Incidentally, AutoDMP¹⁷ used proxy wirelength, congestion, and density costs similar to those proposed in our *Nature* paper, and found that they do in fact correlate with final metrics [5].

4.A.4 Incorrect Claim of Validation by Google Engineers

Cheng *et al.* claimed that Google engineers confirmed its technical correctness, but this is untrue. Google engineers (who were not corresponding authors of the *Nature* paper) merely confirmed that they were able to train from scratch (i.e., no pre-training) on a single test case from the quick start guide in our open-source repository. The quick start guide is of course not a description of how to fully replicate the methodology described in our *Nature* paper, and is only intended as a first step to confirm that the needed software is installed, that the code has compiled, and that it can successfully run on a single simple test case (Ariane).

In fact, these Google engineers share our concerns and provided constructive feedback, which was not addressed. For example, prior to publication of Cheng *et al.*, through written communication and in several meetings, they raised concerns about the study, including the use of drastically less compute, and failing to tune proxy cost weights to account for a drastically different technology node size.

The Acknowledgements section of Cheng *et al.* also lists the *Nature* corresponding authors and implies that they were consulted or even involved, but this is not the case. In fact, the corresponding authors only became aware of this paper after its publication.

¹⁷AutoDMP is also one of the methods compared against in Cheng *et al.*'s Table 1.

Part II

Reinforcement Learning for Language Modeling

Chapter 5

Synthetic Data Generation and Multi-Step Reinforcement Learning

5.1 Introduction

This chapter, the first in Part II, introduces SWiRL (Step-Wise Reinforcement Learning), a synthetic data generation and RL method that improves performance on multi-step reasoning and tool use.

This approach iteratively generates multi-step reasoning and tool use data, and then learns from that data. It employs a simple step-wise decomposition that breaks each multi-step trajectory into multiple sub-trajectories corresponding to each action by the original model. It then applies synthetic data filtering and RL optimization on these sub-trajectories. We evaluated SWiRL on a number of multi-step tool use, question answering, and mathematical reasoning tasks. Our experiments show that SWiRL outperforms baseline approaches by 21.5%, 12.3%, 14.8%, 11.1%, and 15.3% in relative accuracy on GSM8K, HotPotQA, CofCA, MuSiQue, and BeerQA, respectively. Excitingly, the approach exhibits generalization across tasks: for example, training only on HotPotQA (text question-answering) improves zero-shot performance on GSM8K (a math dataset) by a relative 16.9%.

Large Language Models (LLMs) have demonstrated remarkable capabilities in Natural Language Processing [69, 12, 162]. However, they often struggle to answer complex queries that require reasoning and tool use across multiple steps [222], such as multi-hop question-answering, mathematical problem-solving, coding, and other agentic tasks, [226, 202, 222, 49, 94, 57, 126].

Traditional reinforcement learning (RL) approaches, such as RL from Human Feedback (RLHF) [45], RL from AI Feedback (RLAIF) [18], and RL from Execution Feedback (RLEF) [68], have focused on single-step optimization, leaving the challenge of multi-step tasks largely unaddressed. Many real-world problems require a sequence of interrelated actions; for example, when answering a challenging question, a model must determine not just what information to seek, but when to

stop searching and synthesize its findings. Multi-step reasoning creates a compounding challenge, as incorrect intermediate steps often lead to incorrect final results, making it critical to maintain accuracy across the entire chain of actions or learn to effectively recover from such errors.

To address this challenge, we present Step-Wise Reinforcement Learning (SWiRL), an offline multi-step optimization technique. We consider a setting where the model has access to a tool, such as a search engine or calculator, and can run a sequence of tool use calls as needed to answer the question. Our goal is to teach the model how to decompose complex problems into a sequence of more manageable subtasks, when to call the tool, how to formulate a call to the tool, when to use the results of these queries to answer the question, and how to effectively synthesize its findings. In particular, we propose a two stage approach, in which we first generate multi-step synthetic data and then learn from these data using a step-wise reinforcement learning method. This approach has the key practical advantage that we can quickly generate large volumes of multi-step training data via parallel calls to avoid throttling the training process with slow tool use execution. In addition, this offline process enables greater reproducibility due to having a fixed dataset.

To generate multi-step synthetic training data, we provide an open-source LLM (Gemma 2 [71]) with access to a relevant tool (e.g., a search engine or calculator). We iteratively prompt the model to generate multi-step trajectories; at each step, the model is free to generate a chain of thought, and may either call a tool or produce a final answer, which we refer to as the model’s action. If the model generates a tool use call, its query is automatically extracted from the overall response and executed in the environment, and the result is presented to the model in the next step. The trajectory ends when the model generates an answer to the original question, which it indicates using special markers. We convert each trajectory with k actions into k subtrajectories, containing the context from the beginning of the trajectory up to that action. We then use a step-wise reinforcement learning approach to optimize over this dataset, employing a generative reward model that evaluates each action in the context of its subtrajectory.

This granular approach enables us to apply direct feedback after each step of the trajectory, and to do so in a manner that is contextually aware. Unlike prior RL finetuning approaches used in frontier open-source models like DeepSeek-R1 [53] and Llama-3 [77], we do not solely optimize for final performance, and use no golden labels; however, by optimizing for the reasonableness of each step given prior steps, SWiRL does in fact improve final performance.

In addition to evaluating SWiRL on challenging multi-hop question-answering and mathematical problem-solving tasks, we also study the generalization properties of this methodology. This is of key interest because there is an explosion of agentic applications for language models, and methods that generalize across datasets and tasks will be easier, cheaper and faster to adapt to diverse agentic applications of LLMs. We also measure the effectiveness of different synthetic data filtering strategies, study SWiRL’s ability to generalize across datasets and tasks, measure the impact of model size and

dataset size, and explore the mechanism driving these performance improvements.

Our contributions are as follows:

- We propose Step-Wise Reinforcement Learning (SWiRL), an approach to synthetic data generation and offline RL that advances multi-step reasoning and tool use.
- We demonstrate generalization across datasets. For example, training SWiRL on HotPotQA [226] not only improves performance on the dataset itself, but also yields superior performance on other multi-hop question-answering datasets, e.g., 21.5% on GSM8K [49], 15.3% on BeerQA [167], 11.1% on MuSiQue [202] and 14.8% on CofCA [222].
- We also show transfer across disparate tasks, namely mathematical reasoning to question-answering and vice versa. Training only on multi-hop HotPotQA question-answering improves performance on GSM8K [49] (a math dataset) by 16.9%, and training on GSM8K improves performance on HotPotQA (multi-hop question-answering) by 9.2%.
- We analyze the impact of synthetic data filtering strategies in a multi-step reasoning and tool use setting, and demonstrate that models learn best from datasets which have been filtered step-wise to ensure high-quality reasoning traces, but which are not filtered by outcome (correct final answer).
- We explore the impact of training dataset size and model size on SWiRL, observing that significant gains can be achieved even with just 1000 trajectories and that smaller models (Gemma-2-2b and 9b) can benefit from in-domain SWiRL, but do not display the same generalization as their larger counterpart, Gemma-2-27b.
- We demonstrate that SWiRL effectively improves the average process reward, even when evaluated on out-of-distribution tasks, suggesting that the downstream performance gains are driven by improved multi-step reasoning.

5.2 Methodology

Our methodology, Step-Wise Reinforcement Learning (SWiRL), consists of two stages. In the first stage, we generate and filter synthetic data. In the second stage, we use a step-wise reinforcement learning approach to optimize a generative base model on the synthetic trajectories. SWiRL does not require golden labels or human annotations, and instead relies entirely on model-based judgments for data generation, filtering, and RL optimization. The overall flow of our methodology is depicted in Figure 5.2.1 (Stage 1) and Figure 5.2.2 (Stage 2).

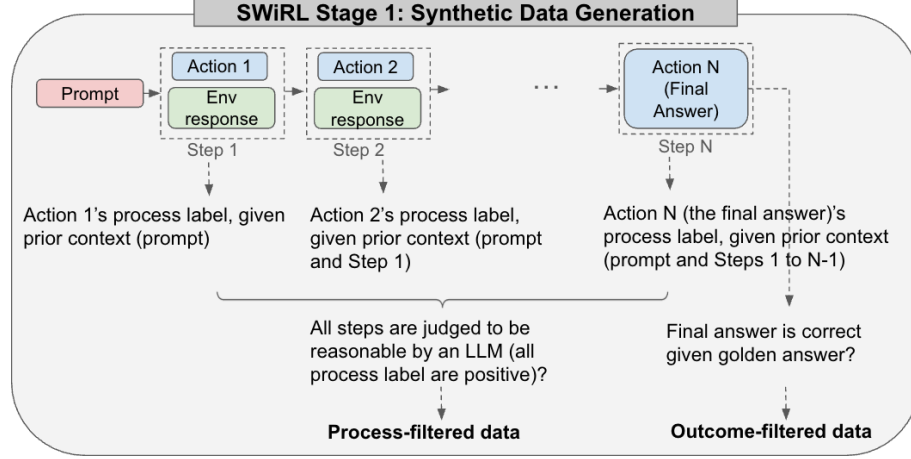


Figure 5.2.1: In SWiRL Stage 1, we generate and filter multi-step synthetic trajectories. At each step, the model is free to generate a chain of thought, call a tool such as a search engine or calculator, and/or produce an answer to the original question. Process-filtered data corresponds to trajectories in which every step is judged to be reasonable by a model judge (Gemini 1.5 Pro Thinking). Outcome-filtered data corresponds to trajectories with a final answer that matches the golden label.

5.2.1 Multi-Step Data Collection

In Stage 1 (see Figure 5.2.1), we generate synthetic trajectories consisting of multiple steps of reasoning and tool use, which we use as training data for the step-wise RL methodology described in the next section. To compile a large-scale collection of synthetic trajectories, we augment a language model with a tool (e.g., a search engine or calculator), and iteratively prompt the model to generate multi-step trajectories. At each step, the model is asked to choose whether to call a tool or produce a final answer, and is always free to generate chains of thought (which it typically does). If the model generates a tool use call, it is parsed from the overall response, executed in the environment, and the result is presented to the model in the next step. See Appendix 5.A for the prompt, which contains a question, explicit instructions regarding multi-step tool utilization, and the results of prior tool use calls.

For each multi-step synthetic trajectory, we define the following annotations. The trajectory itself is denoted by $\tau = (s_1, a_1, \dots, s_K, a_K)$. The first state s_1 is the original prompt. Each following state s_i contains the entire context so far, containing state s_{i-1} , action a_{i-1} , and the environment (tool call) response to a_{i-1} . Each action a_i is the model response, given state s_i . The last action, a_K , is the model’s answer to the original prompt.

In this chapter, we compiled a dataset of 50,000 synthetic trajectories seeded by 10,000 multi-step questions from the HotPotQA training set [226] (i.e., 5 trajectories per question), and a mathematical reasoning dataset of 37,500 synthetic trajectories seeded by the 7,500 questions in the GSM8K training set [49]. Note that, for HotPotQA, we filtered out “Easy” questions, which can typically be answered with a single search query. To prevent synthetic trajectories from being excessively long,

we set a maximum step count of 5 for HotPotQA questions, and 10 for GSM8K questions (which typically require 2-8 steps to solve).

Having compiled these datasets, we consider four different filtering strategies and measure their impact on performance (Figure 5.2.1): (1) No filtering; (2) Process filtering, where we retain trajectories in which each step was deemed reasonable given all previous steps. Concretely, a model (Gemini 1.5 Pro Thinking, in our case) is prompted to render a binary judgment as to whether action a_i is reasonable given the context s_i . See Appendix 5.A for our prompt. No golden labels are used; (3) Outcome filtering, where we select trajectories based solely on whether the final response, a_K , matches the golden answer; and (4) Process and outcome filtering, in which we take the intersection of both filtering approaches and retain only trajectories that exhibit both step-wise soundness and correct final outcomes.

Recent approaches to synthetic data distillation, such as Deep-Seek R1 [53], have demonstrated that synthetic data filtered for correct outcomes can lead to good performance with single-step RL and supervised finetuning (SFT). In this chapter, we sought to explore whether this pattern would hold in a multi-step, tool use setting, and to explore the impact of both outcome and process filters. Like this prior work, we observed that filtering multi-step trajectories for correctness was effective for SFT, and in fact critical for good performance. However, we found that SWiRL, unlike SFT, can learn even from trajectories that end in incorrect final answers. In fact, we achieve our best results by including process-filtered data, regardless of the correctness of the outcome.

5.2.2 Step-Wise Reinforcement Learning Methodology

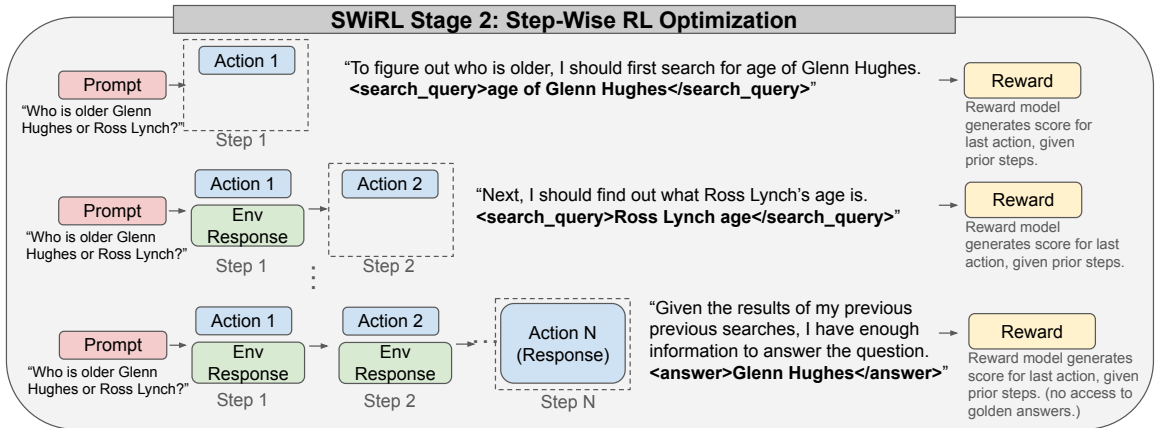


Figure 5.2.2: In SWiRL Stage 2, we perform step-wise RL to train on the synthetic multi-step trajectories from Stage 1. Each step contains an action, which corresponds to a tool call or the final response. The model is free to generate chains of thought during each step. The environment responses are captured in the prior steps of the synthetic trajectories, which were generated offline. Granular feedback is provided by a generative reward model, which is used to perform RL optimization directly on each action, given the prior context.

As shown in Figure 5.2.2, we propose a RL approach capable of learning effectively from the synthetic multi-step trajectories generated in Stage 1. At each step, a base model is optimized to predict either the next intermediate step or the final response based on preceding context. At each step i , the model has access to the full contextual history, including the original prompt, all previous model-generated steps and any applicable environment response corresponding to those steps.

Thus, our objective function is the expected sum of stepwise rewards:

$$J(\theta) = E_{s \sim T, a \sim \pi_\theta(s)} [R(a|s)]$$

Here, π_θ is the base model parametrized by θ , which is finetuned via SWiRL (Note that we also use π_θ to generate synthetic data.) T denotes the set of all states in the synthetic multi-step trajectories, i.e., each incremental state s within each trajectory τ . The reward signal $R(a|s)$ is derived from a generative reward model, specifically Gemini 1.5 Pro in our experiments, which assesses the quality of the generated response a given the context s . No golden labels are used.

We optimize this expected reward using the same policy gradient algorithm used in Gemma 2 to perform RLHF [70, 71]¹ Our granular, step-by-step finetuning paradigm enables the model to learn both local decision-making (next-step prediction) and global trajectory optimization (final response generation) while being guided by immediate feedback on the soundness of each prediction.

5.2.3 Step-Wise Inference-time Evaluation

As shown in Figure 5.2.3, at inference time, we iteratively prompt the model to either call a tool or produce a final answer. If the model generates a search query (indicated by `<search_query>` `</search_query>` tags), we parse out that query, embed it with a Gecko model [125], perform a nearest neighbor lookup in the corresponding vector database, and inject the retrieved article into the model’s context window. If the model generates a calculator tool call (indicated by `<math_exp>` `</math_exp>` tags), we parse out the mathematical expression, execute it with a SymPy interpreter, and inject the calculated results into the context window. This process terminates when the model either produces an answer (signaled by producing `<answer>` `</answer>` tags) or reaches the maximum number of queries (5 for question-answering datasets, and 10 for mathematical reasoning datasets). See Appendix 5.D for example trajectories.

¹This particular RL algorithm remains unpublished, an unfortunate circumstance that I came to realize only after having run all of the experiments in this chapter. For the purposes of this chapter, I think it is sufficient to say that it is a policy gradient method that closely resembles GRPO [185], as the contributions here are largely orthogonal to the choice of RL optimization algorithm.

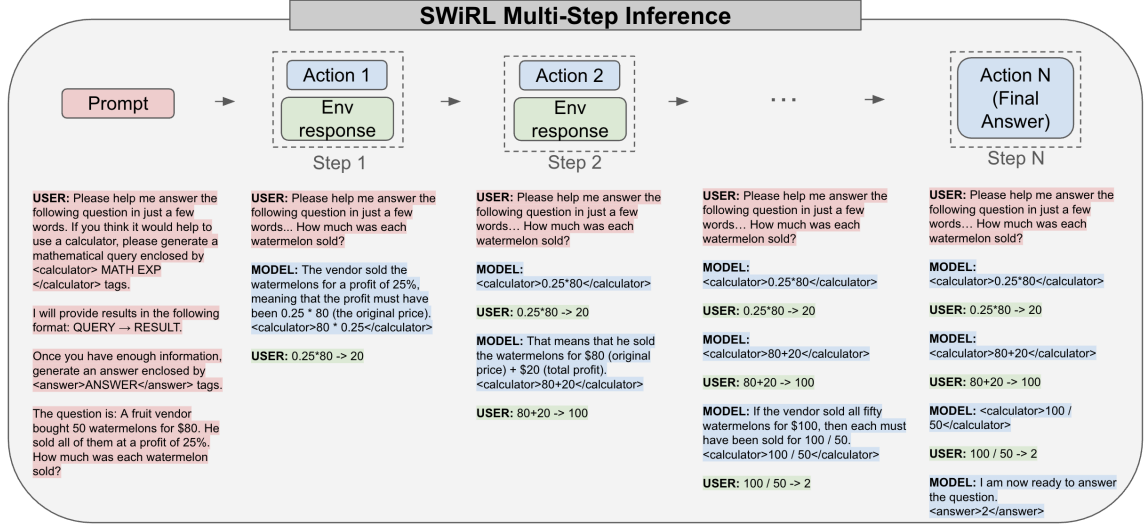


Figure 5.2.3: SWiRL Multi-Step Inference. At inference time, we iteratively prompt the model to call available tools as many times as necessary (up to a limit) before answering the original question. Here, prompts are truncated for clarity and to adhere to space limitations, but full trajectories are available in Appendix 5.D. Note that, in this particular example, the original GSM8K question contains a grammatical error and should read “How much was each watermelon sold for?”.

5.3 Related Work

Reinforcement Learning for LLM Finetuning. One prominent approach, Reinforcement Learning from Human Feedback (RLHF) [164, 45], consists of training a reward model on human preference labels at the response level, followed by RL optimization using Proximal Policy Optimization (PPO) [179]. Building upon this framework, Reinforcement Learning with AI Feedback (RLAIF) [18] has emerged as a scalable alternative that leverages AI models to generate feedback based on predefined principles or constitutions, reducing the need for costly human annotations. RL from Execution Feedback (RLEF) [68] uses environment feedback, such as pass rate on coding test cases, to calculate the reward, which it then optimized via PPO. Besides PPO, other RL optimizations, such as Direct Preference Optimization (DPO) [169] and its successors (e.g., [17, 59, 143, 121]) as well as GRPO [185] have also proven to be effective for finetuning LLMs to maximize a target reward. A limitation of the above approaches is that they focus on single-step optimization with the reward being calculated only at the end of the episode, leading to suboptimal performance for multi-step optimization [133, 213]. In SWiRL, we focus on scenarios where multiple steps of reasoning and tool calls are necessary prior to generating a response. Unlike the above methods, SWiRL enables the model to receive feedback on its granular stepwise actions which leads to better multi-step reasoning and tool use across longer horizons.

Multi-Step Optimization with RL. Recent work including DQO [133] and OREO [213] propose offline reinforcement learning to improve multi-step reasoning for LLMs. However, neither focuses

on enhancing a model’s ability to use tools or interact with an external environment. Additionally, unlike our approach, which optimizes at the (reasoning) step level, DQO relies on token-level actions, which as shown in [213], are generally less effective than step-level actions. Moreover, OREO requires training a separate value network and policy network, and relies on iterative co-optimization of both models. The process of maintaining, training, and serving these two models can be prohibitively expensive, particularly for larger models. PRIME [51] proposes an online approach to improve multi-step reasoning, but does not enable tool use or offline training. Tulu-3 [120] uses verifiable rewards to train a language model to do better at mathematical reasoning, but unlike SWiRL, requires access to golden labels.

Reasoning Improvement with Synthetic Data. Several approaches have been proposed for generating synthetic reasoning data. These methods either rely on golden labels to filter the data or use a combination of golden labels and process or outcome reward models [235, 189]. For example, STaR [235] generates chain-of-thoughts (CoT) for reasoning questions, filters for those that result in correct answers, and performs Supervised Fine-Tuning (SFT) on those reasoning traces. The paper also proposes an augmentation technique called “rationalization”, in which for each question the model answered incorrectly, the model is provided with the correct answer and prompted to generate a CoT that leads to that answer. Rejection finetuning (RFT) [231] is another method that relies on collecting reasoning traces from the model and using those with correct outcomes for SFT. ReST [80] demonstrates strong performance on machine translation by iteratively generating data and then finetuning on that data using either a supervised or reinforcement learning objective. $ReST^{EM}$ [189] is an extension of ReST which outperforms training on human data alone for math and coding evaluations, but which plateaus after a few iterations, presumably due to overfitting. Our method also uses a model-based approach to generate multi-step trajectories. However, we show that using a model to label the steps within each reasoning trajectory leads to higher out-of-domain generalization than using only the trajectories which contain correct final answers, meaning that we do not require golden labels. In addition, we enable the model to use tools iteratively to perform multi-hop question answering and mathematical reasoning.

Process vs. Outcome Based Optimization. There have been a number of attempts to compare the effectiveness of process and outcome-based approaches in the domain of math and reasoning [128, 206, 190]. For example, [128] showed that (Outcome Reward Models) ORMs are more effective than (Process Reward Models) PRMs at the task of ranking samples from a fixed generator model, whereas [206] demonstrated that outcome supervision yields comparable accuracy to process supervision at lower cost, but that the reasoning traces from the resulting model exhibit lower fidelity. Both rely on expensive human annotations and golden labels, and do not explore the differential effect of data

filtering on supervised vs. RL optimization objectives.

5.4 Experiments

Datasets Metrics	HotpotQA PM \uparrow	CofCA (Avg) PM \uparrow	MuSiQue PM \uparrow
Proprietary LLMs			
GPT-4	74.8	51.9	63.9
GPT-3.5	62.8	40.7	53.1
Gemini 1.0 Pro	63.5	33.3	46.9
Bing Chat	72.1	41.6	52.3
o1-preview	76.9	58.5	67.9
Open Source LLMs			
Llama 2-7b	38.5	28.9	34.2
Mistral-7b	34.9	25.6	29.2
Qwen 2-7b	39.3	30.7	33.5
Base Gemma 2-27b	58.6	31.7	35.4
SWiRL Gemma 2-27b (Ours)	67.8	39.3	43.6

Table 5.4.1: Comparison of Accuracy (**PM \uparrow** : Partial Match) across Multiple Datasets: **HotpotQA**, **CofCA** (Average of 2-hop, 3-hop, and 4-hop), and **MuSiQue**. Baseline results were drawn from [222]. The Gemma-2 models, both SWiRL and the base model, were not given access to the context documents, but were allowed to sequentially query a vector database. The SWiRL model was trained on HotPotQA using process-filtered data, and for consistency with baseline results, evaluated on GPT-4o with the same prompts as [222] on 300 randomly subsampled questions. See Appendix 5.E for example ids.

5.4.1 Evaluation Datasets

To evaluate performance on multi-step search tool use, we selected five challenging multi-hop question-answering and mathematical reasoning datasets:

- **HotPotQA** [226] is comprised of multi-hop questions from a variety of domains. Human annotators constructed the questions to be answerable only by combining information from two paragraphs of Wikipedia.
- **MuSiQue** [202] is a multi-hop question-answering dataset constructed by chaining together multiple single-hop questions.
- **CofCA** [222] is a multi-hop dataset constructed to be answerable only by querying a counter-factual version of Wikipedia. It contains 2- to 4-hop questions.
- **BeerQA** [168] is an extension of HotPotQA designed to include an even greater number of hops than the original dataset.
- **GSM8K** [49] is a dataset composed of grade school math word problems, which typically take 2-8 steps to solve.

For question-answering datasets, we set up a vector database containing all articles from each data split using Gecko-1B with 768-dimensional embeddings (English) [125].

For the experiments in Table 1, we follow the same procedure as [222], evaluating performance on 300 randomly subsampled examples from the target dataset, using the same language model as a judge (GPT4o) and the same prompt. For every other experiment in this paper, we used Gemma-2-27b as our judge, as this was more cost effective, with the exception of GSM8K for which we used Gemini 1.5 Pro as it exhibited noticeably better numeric evaluation. Model-based evaluation is emerging as a scalable and less brittle alternative to exact match and F1 metrics [238, 78], but does introduce a new source of stochasticity into the evaluation. See Appendix 5.C for our own manual inspection and error analysis of three different model judges.

As described in Section 5.2.3, for each question, we iteratively prompt the model to either call a tool or produce a final answer, and limit the maximum number of queries to 5 for question-answering datasets, and 10 for mathematical reasoning datasets.

5.4.2 Results and Discussion

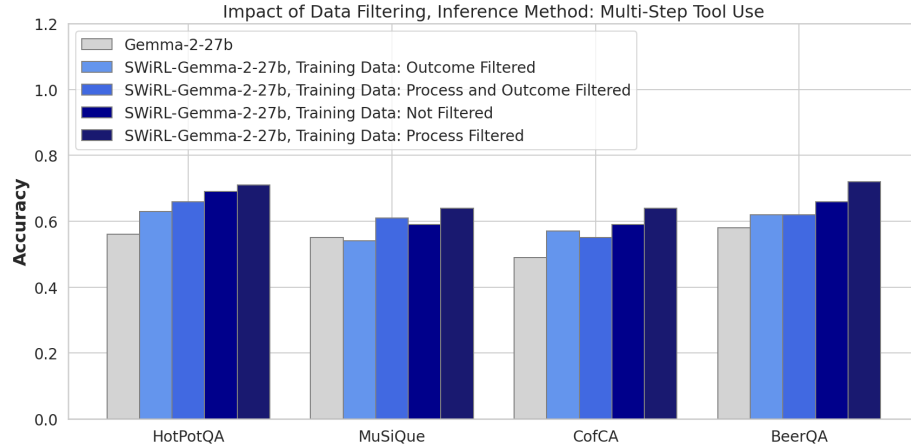


Figure 5.4.1: Impact of Data Filtering on Model Performance. The synthetic trajectories used for training were derived from HotPotQA prompts. SWiRL learns to perform multi-hop question answering even when trained on unfiltered synthetic data. SWiRL’s best performance comes from training on process-only filtered data, where the data is selected based on the soundness of each step within its reasoning traces, but which includes both correct and incorrect responses.

Impact of Data Filtering on Model Performance: We evaluated the influence of various filtering mechanisms on downstream task accuracy, as shown in Figure 5.4.1. Concretely, we consider 4 different types of filtering: no filtering, outcome-based filtering that ensures correct final answers, process-based filtering that ensure that each step is correct as judged by a model, and both process and outcome-based filtering.

In all experiments, we fix the number of trajectories used for finetuning (with the exception of our

	GSM8K (math)	HotPotQA (qa)	CofCA (qa)	BeerQA (qa)	MuSiQue (qa)
Base Model	0.65	0.65	0.54	0.59	0.45
SWiRL on GSM8K (math)	0.79	0.71	0.56	0.68	0.49
SWiRL on HotPotQA (qa)	0.76	0.73	0.62	0.68	0.50

Table 5.4.2: SWiRL Generalization Performance. Finetuning on synthetic traces from HotPotQA or GSM8K improves performance on both in-distribution and out-of-distribution tasks. Interestingly, training on a different domain and tool (e.g., math and a calculator) improves performance on question-answering with a search engine and vice versa, suggesting the effectiveness of SWiRL in improving general multi-step reasoning and tool use capability.

ablation study on the impact of scaling dataset size), and we provided all models with access to an appropriate tool. Notably, process-only filtering consistently yields the highest accuracy, suggesting that focusing on the procedural aspects of data refinement is more important than the correctness of a training trajectory. While both unfiltered and filtered data demonstrated an improvement over the baseline model, filtering for correctness usually harms performance; with the exception of MuSiQue, outcome-filtered or outcome and process-filtered data is less effective than unfiltered data. We hypothesize that this is because SWiRL actually benefits from having access to both positive and negative examples. These results underscore the relative unimportance of outcome-based filtering, which requires golden labels. They also demonstrate that our process RL method can effectively learn from even trajectories with incorrect final answers.

Generalization Across Disparate Tasks: To measure generalization across training tasks, we evaluated the mathematical reasoning capabilities of a model trained on multi-hop question-answering with search tool use (HotPotQA). Specifically, we evaluated the performance of this model on GSM8K, a mathematical reasoning task, providing the model with a SymPy interpreter to use as a calculator. This experiment was run on a different random subsample of 300 examples. As shown in Table 5.4.2, applying SWiRL on out-of-distribution data and tasks still improves performance.

Comparison of Supervised Finetuning and SWiRL: Figure 5.4.2 compares the performance of Supervised Fine-Tuning (SFT) and SWiRL on downstream tasks. The results show that SFT leads to worse overall performance when compared to SWiRL across all data filtering strategies. We observe that SFT performs better if we apply it to data that is both process and outcome-filtered, rather only process-filtered. However, interestingly, SWiRL learns best from data that is only process-filtered. We attribute this to SFT’s tendency to memorize, rather than generalize [46, 181], which can hinder the model’s performance on new, unseen scenarios. In contrast, SWiRL has the ability to improve model performance by targeting per-step reward maximization. SWiRL enables the model to develop a deeper understanding of the necessary steps of query generation and retrieval, which leads to

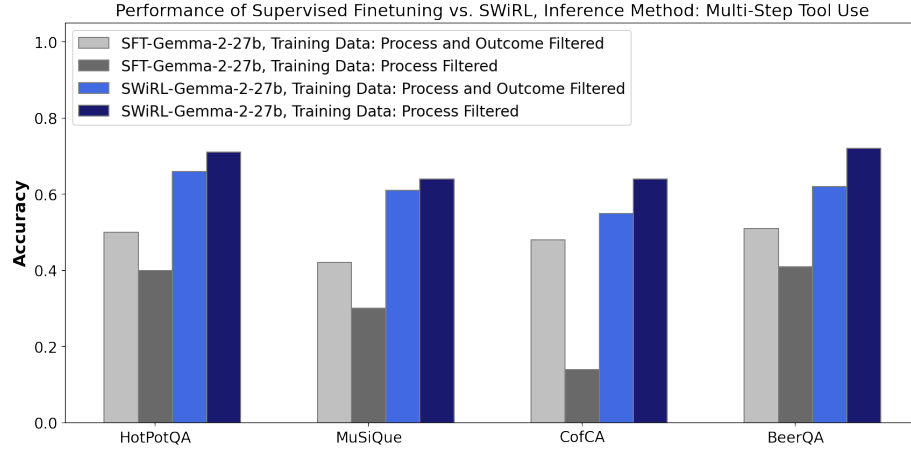


Figure 5.4.2: Comparison of SFT and SWiRL. Synthetic data for training is derived from HotPotQA, and to derive accuracy, Gemma 2 27b evaluates whether the model’s answer matches the golden answer. SWiRL greatly benefits from process-only filtered traces, and unlike SFT, is capable of learning from traces with both correct and incorrect outcomes.

enhanced planning and generalization.

Effect of Tool Use: As discussed in Section 5.2.3, at inference time, we use the proposed multi-step eval as shown in Figure 5.2.3 and we iteratively prompt the model to make tool calls as necessary to answer the question. As shown in Figure 5.4.3, both base and SWiRL models improve with SWiRL’s multi-step tool use inference, but SWiRL-training offers even further improvements. Without access to a vector database, SWiRL does not outperform the base model, likely reflecting that these questions cannot be answered without access to particular facts. However, the SWiRL model exhibits substantial improvements in mathematical reasoning (GSM8K), even without access to a calculator tool, suggesting that SWiRL training may improve the model’s ability to break down complex problems into multiple manageable subtasks. However, the results on multi-hop question-answering without tool use are more mixed,

Impact of Scaling Finetuning Dataset and Model Size: Our experiments on scaling the fine-tuning dataset size reveal a clear trend: SWiRL has the ability to leverage larger datasets, even when using only process-filtered data, as shown in Figure 5.4.4. As the fine-tuning dataset size increases, a consistent enhancement in model performance is observed across our target multi-step reasoning tasks. While a limited dataset of 100 data points appears insufficient for the model to effectively generalize, a significant improvement is evident with 1,000 data points, showing solid gains across all datasets. Furthermore, scaling up to 10,000 data points continues to yield further performance enhancements, confirming the efficacy of our method in capitalizing on larger datasets for improved reasoning capabilities.

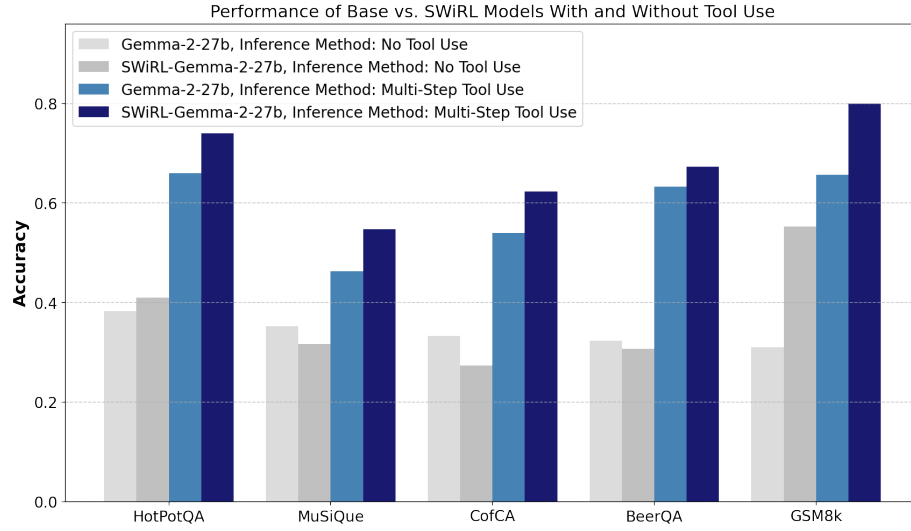


Figure 5.4.3: Performance of SWiRL with and without multi-step tool use. SWiRL’s multi-step tool use inference improves the performance of both the base model and the SWiRL-finetuned model, but benefits the latter substantially more. Without access to a calculator tool, the SWiRL model outperforms the base model on mathematical reasoning, but performance on multi-hop question-answering suffers, as answering knowledge-based questions is more dependent on knowledge of particular facts and therefore access to the search tool.

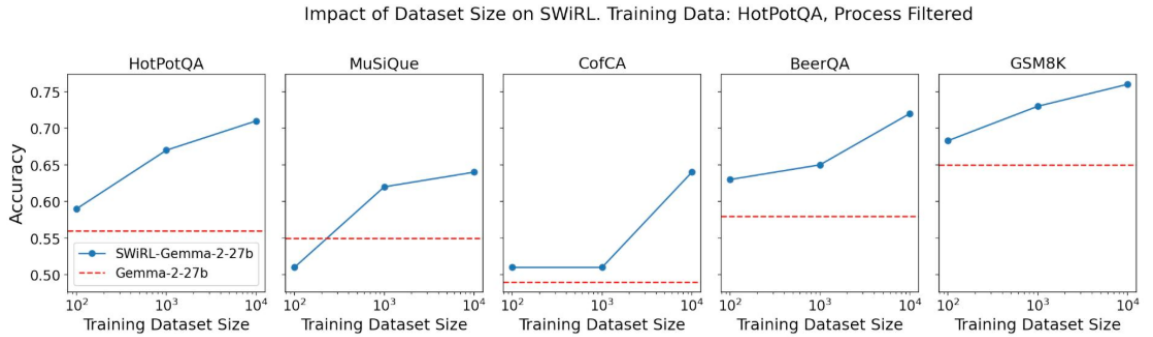


Figure 5.4.4: Performance as a Function of Synthetic Dataset Size. Synthetic training data is derived from HotPotQA, and accuracy is evaluated by Gemma 2 27b. As we scale the dataset size, we observe consistent improvements in model performance. With only 1000 data points, the model robustly improves both on in- and out- of distribution datasets.

We also varied model size, observing that smaller models (2b and 9b) may benefit from in-domain SWiRL, but do not display the same generalization as their larger counterpart, Gemma 2 27b. See results in Appendix 5.B.

Effect on Mean Process Label Accuracy:

In the previous subsections, we evaluated the effect of SWiRL on downstream task accuracy. Here, we take a deeper look to understand how SWiRL achieves these performance improvements. In Table 5.4.3, we show the average process label accuracy for the baseline model vs. a SWiRL finetuned model on 500 trajectories (seeded by 100 questions) for both HotPotQA and GSM8K. To calculate the score per step, we use the same model and prompt as we used for process filtering, as described in Section 5.4.1. We take a macro-average of the process label scores within and then across trajectories. We observe that both for in-distribution and out-of-distribution tasks, the SWiRL model generates trajectories with higher average process labels, suggesting that the higher final accuracies are driven by better multi-step reasoning.

	HotPotQA (in distribution)	GSM8K (out of distribution)
Base (Mean Process Label)	82.5%	87.5%
SWiRL on HotPotQA (Mean Process Label)	91.0%	91.6%

Table 5.4.3: Impact of SWiRL on Process Correctness. After our multi-step RL optimization, we observe that the average correctness of each step improves over the base model on both in- and out-of distribution tasks.

5.5 Conclusion

In this chapter, we propose a synthetic data generation and offline reinforcement learning approach to multi-step reasoning and tool use. This approach outperforms baselines by an average 15% across challenging multi-hop question-answering and mathematical reasoning tasks. We explore the effect of different data filtering strategies in a multi-step, tool use setting, and find that our RL approach is effective even on unfiltered data, but performs best on process-filtered data. Unlike supervised finetuning, our RL approach can learn from trajectories with incorrect final answers and actually benefits from the presence of a mixture of both correct and incorrect final answers. SWiRL demonstrates strong generalization properties, improving performance on mathematical reasoning (GSM8K) by 16.9% when trained on multi-hop question-answering (HotPotQA) and 9.2% vice versa.

Appendix

5.A Prompts for Synthetic Data Generation, Filtering, and Evaluation

In this chapter, we use the following prompts for data generation, filtering, and evaluation.

Prompt Type	Prompt Text
Prompt for Multi-Step Synthetic Data Generation for Question-Answering with Search Tool Use	<code><start_of_turn>user</code> Please help me answer the following question in just a few words. If you think it would help to do a search, please generate a search query enclosed by <code><search_query></code> QUERY <code></search_query></code> tags. Some questions may require multiple searches in order to answer, so I will allow you to make up to {} sequential queries before answering the question. Please do not repeat queries you have already issued, as this is a waste of time. I will provide search results in the following format: QUERY → RESULT. Once you have enough information, generate an answer enclosed by <code><answer></code> ANSWER <code></answer></code> tags. Please either issue a search query or answer the question, but not both. The question is: {} <code><end_of_turn></code>

Prompt Type	Prompt Text
Prompt for Multi-Step Synthetic Data Generation for Mathematical Reasoning with Calculator Tool Use	<p><start_of_turn>user</p> <p>Please help me answer the following question in just a few words. If you think it would help to use a calculator, please generate a mathematical query enclosed by <math_exp> MATH EXP </math_exp> tags.</p> <p>Some questions may benefit from using a calculator multiple times in order to answer, so I will allow you to make up to {} sequential queries before answering the question.</p> <p>Please do not repeat queries you have already issued, as this is a waste of time.</p> <p>I will provide results in the following format:</p> <p>QUERY → RESULT.</p> <p>Once you have enough information, generate an answer enclosed by <answer>ANSWER</answer> tags.</p> <p>Please either issue a search query or answer the question, but not both.</p> <p>The question is: {}</p> <p><end_of_turn></p>

Prompt Type	Prompt Text
Prompt for Process-Filtering on Multi-Step Search Tool Use Trajectories	<p><start_of_turn>user</p> <p>My boss asked me to answer the following question with the help of a search engine: {}</p> <p>This means that I might need to decompose the question into a sequence of searches before being able to answer the question.</p> <p>I am trying to learn how to do this more effectively, so please provide feedback on my last message.</p> <p>Please take a look at our conversation so far: {}</p> <p>When evaluating a message, please only consider the last message and do not penalize or reward me for previous messages.</p> <p>When evaluating an answer, please consider only whether the answer follows from the search results, and not whether you believe the answer to be correct.</p> <p>If there is not enough information from the search results to answer the question, you should rate any answer as "BAD". Pay close attention as it may initially seem like the answer is present when it is not.</p> <p>When evaluating a search query, please consider whether it is likely to help me answer the original question.</p> <p>Explain your reasoning and then answer with either "GOOD" or "BAD".</p> <p><end_of_turn></p>

Prompt Type	Prompt Text
Prompt for Evaluation / Outcome-Filtering on Multi-Step Trajectories with Search Tool Use	<p><start_of_turn>user</p> <p>I need you to help me grade the answer to the following question: "{}".</p> <p>The answer key says: {}, and my answer is {}. Am I correct?</p> <p>Please explain your reasoning and then answer "YES" or "NO".</p> <p>Do not use your own knowledge to the decide, but simply check whether I gave the answer in the answer key.</p> <p><end_of_turn></p>

Prompt Type	Prompt Text
Prompt for Evaluation / Outcome-Filtering on Multi-Step Trajectories with Calculator Tool Use	<p><start_of_turn>user</p> <p>I need you to help me grade the answer to the following question: "{}".</p> <p>The answer key says: {}, and my answer is {}. Am I correct?</p> <p>Please explain your reasoning and then answer "YES" or "NO".</p> <p>There are multiple ways to write the same answer. For example, "10", "10.00", "\$10", and "\$10.00" are all equivalent.</p> <p><end_of_turn></p>

5.B Impact of Model Size on Effectiveness of SWiRL

The trend is that models are growing in parameter count over time [182], so measuring the impact of model size on the effectiveness of a method can provide insight into its longevity and future impact. It is also interesting to see whether larger models are able to learn more general patterns from the training process, and therefore exhibit greater transfer learning across datasets and even domains (e.g., math vs. question-answering). As shown in Figure 5.B.1, SWiRL demonstrates a clear performance boost over the baseline Gemma 2-27b model, showcasing consistent improvements across both in-domain (HotPotQA) and out-of-domain datasets (MuSiQue, COFCA, and BeerQA); while the 2b and 9b Gemma models also exhibit enhanced performance on in-domain data, their generalization performance on out-of-domain data is less consistent. This suggests that the effectiveness of SWiRL grows with increased model size, which is consistent with the observation that methods such as RLHF [164] and RLAIFF [18] are more effective for larger models.

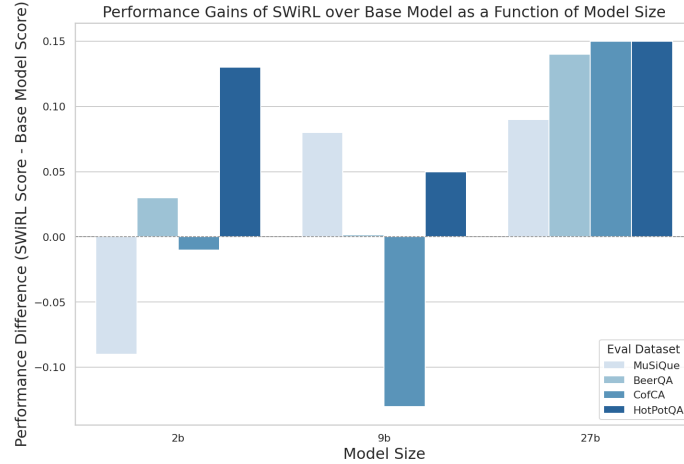


Figure 5.B.1: SWiRL Performance vs. Model Size. Synthetic data for training is derived from HotPotQA. Step-Wise RL finetuning robustly improves performance over baseline for the 27b model across both in-domain (HotPotQA) and out-of-domain datasets (MuSiQue, CofCA, and BeerQA). However, while the in-domain improvements hold for smaller models, the out-of-domain performance is mixed, suggesting that the relative effectiveness of SWiRL is higher for larger models.

5.C Error Analysis of Three LLM Judges

Table 5.C.1: Error Rates for Gemma-2-27b Judgments on HotPotQA (N=100)

Metric	Rate (%)
False Positive Rate (FPR)	4
False Negative Rate (FNR)	1

Table 5.C.2: Manual Analysis of LLM Math Grading Accuracy (N=100)

Model	FP	FN	Notes
Gemma-2-27b	15	0	Overly permissive; all errors involved units.
GPT-4o	0	10	Overly harsh; all errors involved units.
Gemini 1.5 Pro	4	0	Accurate, slightly permissive; all errors involved units.

To evaluate the suitability of language models to serve as evaluators (i.e., check the correctness of a model answer, given a golden answer), we manually checked the correctness of 100 model judgments from Gemma-2-27b on HotPotQA questions. As shown in Table 5.C.1, we found that the error was relatively low (4% false positives and 1% false negatives), justifying the use of this low cost open-source model as our LLM judge.

However, we noticed that Gemma-2-27b made more errors when it came to numeric quantities, so we decided to run a separate analysis for GSM8K, manually evaluating 100 model judgments each for three language models (Gemma-2-27b, GPT-4o, and Gemini 1.5 Pro). Interestingly, we found that Gemma-2-27b tended to be overly “permissive” in its grading, but had zero false negatives, whereas GPT-4o had a relatively high false negative rate but no false positives. We also observed that relative results were consistent across model judges; if GPT-4o gave a higher accuracy score to a particular model, Gemma-2-27b did as well, even if the absolute scores differed. To reduce noise, we chose to use Gemini 1.5 Pro as the LLM judge for GSM8K, in spite of its higher cost.

5.D Example Trajectories

For clarity, we provide example trajectories from question-answering and mathematical reasoning datasets, which showcase both search and calculator tool use.

Trajectory Type	Trajectory Text
Example Multi-Hop Question-Answering Trajectory with Search Tool Use Generated by SWiRL model on HotPotQA	<p><start_of_turn>user Please help me answer the following question in just a few words. If you think it would help to do a search, please generate a search query enclosed by <search_query> QUERY </search_query> tags. Some questions may require multiple searches in order to answer, so I will allow you to make up to 5 sequential queries before answering the question. Please do not repeat queries you have already issued, as this is a waste of time. I will provide search results in the following format: QUERY → RESULT. Once you have enough information, generate an answer enclosed by <answer>ANSWER</answer> tags. Please either issue a search query or answer the question, but not both. The question is: What company published both The Scorch Trials and The Death Cure? <end_of_turn> <start_of_turn>model <search_query>the scorch trials publisher </search_query> <end_of_turn> <start_of_turn>user the scorch trials publisher -> The Scorch Trials is a 2010 young adult post-apocalyptic dystopian science fiction novel written by American author James Dashner and the second book, fourth chronologically, in "The Maze Runner" series. The novel was published on September 18, 2010 by Delacorte Press. It is preceded by "The Maze Runner", and followed by "The Death Cure". A was released on September 18, 2015 by 20th Century Fox. <end_of_turn> <start_of_turn>model <search_query>The Death cure publisher </search_query> <end_of_turn> <start_of_turn>user The Death cure publisher -> The Death Cure is a 2011 young adult dystopian science fiction novel written by American writer James Dashner and the third book, fifth chronologically, in the "Maze Runner" series. It was published on October 11, 2011 by Delacorte Press and was preceded by "The Maze Runner" and "The Scorch Trials" and followed by the series prequels, "The Kill Order and The Fever Code." <end_of_turn> <start_of_turn>model <answer>Delacorte Press</answer> <end_of_turn><eos></p>

Trajectory Type	Trajectory Text
Example Multi-Step Math Reasoning Trajectory with Calculator Tool Use Generated by SWiRL model on GSM8K	<p><start_of_turn>user Please help me answer the following question in just a few words. If you think it would help to use a calculator, please generate a mathematical query enclosed by <math_exp> QUERY </math_exp> tags. Some questions may benefit from using a calculator multiple times in order to answer, so I will allow you to make up to 10 sequential calls before answering the question. Please do not repeat queries you have already issued, as this is a waste of time. I will provide search results in the following format: QUERY → RESULT. Once you have enough information, generate an answer enclosed by <answer>ANSWER</answer> tags. Please either issue a math query or answer the question, but not both. The question is: Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May? <end_of_turn> <start_of_turn>model <math_exp>48 / 2 </math_exp> <end_of_turn> <start_of_turn>user 48 / 2 -> 24.0 <end_of_turn> <start_of_turn>model <math_exp>48 + 24</math_exp> <end_of_turn> <start_of_turn>user 48 + 24 -> 72.0 <end_of_turn> <start_of_turn>model <answer>72</answer> <end_of_turn><eos></p>

5.E Example Ids

For reproducibility, we provide the ids of the examples used for evaluation.

HotPotQA Example Ids

```
5ab46a295542990594ba9c1c, 5a85ad925542997b5ce40023, 5a8353285542996488c2e40d, 5a79be835542994bb9457052,
5a82a0fc55429954d2e2eb87, 5adc2c9355429947ff173904, 5ab1e71b554299340b525421, 5a7790ac5542992a6e59def9,
5a83e4195542990548d0b243, 5ab3239b554299194fa93574, 5ae1a460554299234fd042a8, 5a81e07554299676ccbe128,
5a7f714c5542992097ad2f6e, 5ab639c055429953192ad2aa, 5a7c1fe4554299683c1c62cf, 5ab7cf3554299281fe391e,
5aba5b2455429939ce03dc9c, 5a7173b45542994082a3e83c, 5a90049d55429933b8a20468, 5a8e0d7e5542995085b373b4,
5adfbf3155429906c02daa29, 5abf1fed5542990832d3a127, 5addf6415542990dbb2f7f25, 5a8138c155429938b6142300,
5a7e77b554299042af8f6b0, 5ae293fb5542996483e649fe, 5ae40a8b55429970de88d8a9, 5ab457445542991751b4d748,
5a77d605542995d83181301, 5a89c2715542993b751ca990, 5a7a4d845542990783324f04, 5ae0fef55542990adbac6fdf,
5a72321f55429971e9dc934a, 5ac440355542995c82c4ad0d, 5a7dd8625542990b8f503ae8, 5ab48dd55542991779162cd9,
5abc3948554299700f9d782b, 5a8a7cb255429930ff3c0df8, 5ae1178e5542997b2e7fd0d6, 5abd08ae554299700f9d7980,
5ab1e5975542997061209590, 5a74dca85542996c70cfae1f, 5ab8348d55429934fafe6d13, 5a78f1ef55429974737f7919,
5ac16eb355429964131be1f5, 5ae5e12d55429929b08079e4, 5ade6bbf5542997c77adee24, 5adf573c55429955348e798,
5a8901d9554299515336125b, 5a89fd9e55429970aeb701e8, 5a7917d955429974737f7982, 5adc1017554299438c868d20,
5a8da5c355429941ae14ddf, 5a8cad265542996e8ac88b19, 5add4ae25542992200553a88, 5ae026eb55429924de1b703a,
5a74fcb5542996c70cfae67, 5adfa8ac55429942ec259add, 5adb4f555542994650320c18, 5ac31609554299741d48a1c0,
5a7b65bf55429931da12ca86, 5a73870455429905862fe051, 5a8b009755429950cd6af4c0, 5ae62b2d5542992ae0d162b5,
5a7bd5795542992d025e6825, 5ab3185755429976abd1bc5f, 5ac046475542996f0d89cb70, 5a89138255429951533612af,
5a85d69f5542997175ce2062, 5a82dfa455429940e5e1a938, 5a8730355542991e7718170f, 5a85b34a5542994c784ddbd4d,
5a8658c4554299211dda2b02, 5abd9fa55542996e802b4809, 5ab268aa5542993be8fa9908, 5ae5dce755429929b08079d8,
5a727ef15542992359bc30c5, 5a8e2ba85542995a26add474, 5a84f9465542991dd0999e36, 5a87099455429960ec39b704,
5a864d835542994775f6073c, 5ab9bf3b554299743d22ebeb, 5a864dfc5542994775f6073f, 5a871ce055429960ec39b749,
5a8bd3375542997f31a1dd3, 5ab277965542993be8fa9919, 5abcea83554299114383a194, 5a8975615542992815336130b,
5adfd44a55429906c02daa7c, 5ae265bb5542992decdbccca, 5a84b3035542992a431d1a91, 5a77280b5542994ace3b71ff,
5ae4441355429908b6326488, 5a76de035542994aec3b718d, 5a7d2045554299452d57bb09, 5abc7af15542993a06ba8fed,
5abdddeb5542991f66106083, 5a8218855542990a1d231f4e, 5a732fb5542992359bc3271, 5a8024ad5542992097ad2fde,
5ae142a4554299422ee9964a, 5a72d5155542991f9a20c5b4, 5a722a4b55429971e9dc931f, 5a7a9eca55429941d65f26f3,
5adfa5405542992d7e9f93ca, 5a7b78c3d55429927d897bfec, 5a7c6ac25542996dd594b925, 5abae9cd5542996ce54e9f04,
5ae182a455429934fd0428f, 5a84d29d5542994c784dda60, 5ae44eeb5542995dadf2430f, 5adb6e7b455429976d4830ac2,
5abedd105542993f9e9a41d63, 5a80a7df554299485f59867f, 5ab2f6b1554299545a2cfaea, 5ac29ddc554299657fa28dfc,
5a7222ce55429971e9dc92c7, 5ae221f15542994d89d5b366, 5a7f9cc25542995d8a8dded2, 5abed42a55429976d4830ac2,
5ac329e45542991a06ce993e, 5a882caa5542997ce5c09a596, 5ac1a94455429964131be262, 5a762e0f5542992d0ec06052,
5a7918ec554299148911f9ef, 5a7e0bd25542997ce2c4750b, 5a8af3fc55429916710eb0ac, 5aba94465542994dbf019953,
5a82eff725542995ce29dcd0a, 5ab2a5fb554299545a2cf9ef, 5ab3dd4ae5542992ade7c6ec5, 5ac25882554299636651998c,
5a535f55542993aec5ec17c, 5ac55c915542993c66e8234f, 5adfcf7655429906c02daa49, 5a8a1255542992e4fca84f1,
5a8a8f82c55429950cd6af31, 5a8c564b554299240d9c2128, 5a89efb25542992e4fca8497, 5ab58009554299637185c5b2,
5ae69a455542996d980e7c48, 5a8f8dfb5542997ba9cb32bb, 5a811e1955429903bc27b931, 5a8172955542990a1d231ee,
5ad428955429959677d6a67, 5ac263a25542992f1f2b38a3, 5ac5190d5542996feb3fe9f8, 5a82bfbc55429954d2e2eb5e,
5abce73b5542993a06ba9a2, 5adb6f72554299438c868cf0, 5a75dd02554299109176e5aa, 5a8200d0554299261cdad2e,
5a8090105542996402f6a55c, 5adfd3a36554299025462a35e, 5a7f9e0155429969796c1ae, 5a7b5f64554299042a8f757,
5a8a7bf55542996c9b8d5eff, 5a7f3fae5542991bbcb971c9, 5a77b0795542992a6e59df89, 5ac178655542994ab5c67d5a,
5ab5eab35542992aa134a3dd, 5ab667be55429954757d328a, 5a7a333f5542996a35c17130, 5ac262a055429951e9e6859a,
5a7eae9d5542994846c1cdc6f, 5ac1985e55429964131be248, 5a848c215542992a431d1a4f, 5a89a79c5542993b751ca970,
5ae16d355429917b4a5bd18, 5a7289755542992359bc30d9, 5a7d1dd055429909bec76960, 5ac152e755429964131be1bb,
5ae74df4554299540e5a5659, 5ac21559554299492dc91bc2, 5a8935e6554299669944a506, 5a831cb955429966c78a6b3f,
5a77aa555542992a6e59df6a, 5abff5e95542997d6429596a, 5ae07634554299603e418412, 5ab4eb2b55429942dd415fa2,
5abd512655429924427fcfb4, 5a7ad0195542992d025e66fd, 5a7cf9b455429907fabef07c, 5ae0fa52554299422ee99594,
5ae24d1a5542992decdbcca6, 5a7144df5542994082a3e72f, 5ac0279c5542996f0d89cb3f, 5a8a93ac55429948461cead,
5adec595542992fa25da83f, 5abbfd00554299114383a0d4, 5a7b9cac554299042af8f78f, 5ab9020d5542991b5579f0ca,
5a7c1c595542990527d55456, 5a7c583e5542996dd594b910, 5a8e72f05542990e94052b13, 5a85a1015542991dd0999e6f,
5adcb8205542994ed6169bd2, 5a8cef7a554299441c6b9f8a, 5a7fee435542994857a7685b, 5a7b4f2c55429931da12ca66,
5abeaf8a5542997ec76fd346, 5abbbe7e5542993f40c73c05, 5a8f4e8955429918e830d1f1, 5ac1a0e15542994ab5c67dab,
5a7a9b4755429941d65f26ef, 5a87c1ac5542997e5c09a565, 5ab962ff554299131ca4231f, 5a7b79c95542997c3ec971b0,
5abe3ac35542993f32c2a0ac, 5a7639d55542992db9473748, 5a7a2ec05542990198eaf0bc, 5ac3d31a5542995ef918c249,
5abac3eb5542996cc5e49ee2, 5adff38b55429925eb1afb7d, 5ab7530b55429928e1fe3849, 5a88dcef55429938390d3fe3,
5ae0027b55429942ec259bda, 5a85ec815542994775f606af, 5ac172a15542994d76dcce2e, 5ac073be5542996f0d89cb8d,
5ae5262755429924173fb60f, 5a8e72fe5542990e94052b14, 5a76133755429976c32bcff, 5ae6b38c5542992ae0d16392,
5ab98fee554299131ca4237c, 5ac0e564554299294b219045, 5a72ede5542992359bc31da, 5a7b663355429931da12ca87,
5a7cbe0f55429909bec767ee, 5a845bdd5542996488c2e524, 5a8a28b55542996c9b8d5e23, 5ae5fb975542996de7b71aa8,
5aba9cf5542994dbf01997e, 5ae11f0b5542997b2ef7d0e0, 5abe16c655429976d4830a71, 5abdbdd35542992cdd8e7fc6,
5abedbf5542993fe9a41d5f, 5a792421554299148911fa09, 5a80c5f6554299260e20a151, 5ab4136b5542996a3a969f18,
5adc375055429944faac246c, 5ac14d9d55429964131be1ab, 5abf23a65542997ec76fd3d7, 5a7e1d4255429965cec5ea79,
5ae63c8f5542992663a4f27c, 5ae71816554299572ea546d1, 5ae4db55429913cc2044ee, 5ae4a09e5542996836b02ced,
5ac2312755429964131be2c3, 5ae36d325542992c3233c3f8, 5a7d68045542995f4f0226d, 5aba88d555429901930fa811,
5ae1e4b554299068b959e63, 5a7e6d325542991319bc94a7, 5ab96d865542996be20204df, 5ae4d2c255429960a22e01f6,
5a8053cf5542992097ad2f0, 5a8db1b75542994ba4e3dd01, 5a8d40c95542994ba4e3dc3b, 5ae5af10554299546b8f2f23,
5a8d48ff5542994ba4e3dc5a, 5ab5f694554299488d4d9a66, 5a8f99bc5542991e830d28d, 5add0ed35542990d50227dac,
5ac838235542995e66a4755f, 5ab6ccf155429954757d3372, 5ae44fe75542995dadf24314, 5adcb67e5542994ed6169bca,
5abe833d5542993f32c2a140, 5a8b002155429950cd6af3c, 5a76f3c65542994aec3b719a, 5ab5207c5542996a4a36a02b,
5a8a73dd5542996c9b8d5eee, 5a9063c955429933b8a2050f, 5a7b45c855429931da12ca4a, 5a8e8b6c5542990e94052b43,
5a7a57935542990783324f1d, 5abe225c5542991f661060ec, 5a72a6b65542994cef4bc3b7, 5ab7f3625542995dae37ea06,
5a7cfdad55429907fabef095, 5a8994505542993b751ca950, 5ac308775542992decdbcdcd, 5ab72f32554299110f219ac3,
5a7b93ce5542995eb53be961, 5a88710b554299206df2b26b, 5ab6259855429953192ad272, 5ac29ca6554299218029dac0,
5ac0ab335542992a796ded5d, 5ade469c5542992fa25da722, 5ab318a0554299233954ff07, 5ab1f75d554299340b525443,
5ade564554299728e26cd5, 5ae4a3b65542995ad6573dee, 5ae40e3955429970de88d8c5, 5ab9025855429934fafe6e47,
5a82100955429926c1cdae1e, 5ac5138c5542994611c8b36a, 5ab2eb7755429929539468b9, 5ab738945542993667793f97
```

CofCA Example Ids

5a866fee5542991e77181657, 5a7db2f75542990b8f503a34, 5a8ee0a35542990e94052ba0, 5ae525835542990ba0bbb1cd, 5ac4bfd05542997ea680caab, 5ac4c61a5542996feb3fe93c, 5abbbd0f55429931dba144d5, 5a89372855429951533612e6, 5ab381b155429969a97a816b, 5ac2a912554299218029dae8, 5ae3345f55429928c4239682, 5a7bb3d9554299294a54aaa0, 5abaa25155429901930fa868, 5ac39a1c554299657fa290f9, 5a7332b5542992359bc3287, 5ae655c855429908198fa599, 5add82fc5542997545bbdb57, 5add117e5542990d50227db2, 5ab93287554299753720f78f, 5ab979da554299131ca4233a, 5a79c7f95542994bb9457099, 5ab58ae15542992aa134a357, 5ae3bdfa5542990afbd1e1c0, 5add7d055542990dbb2f7e61, 5ae136f655429920d5234325, 5a80b4635542992bc0c4a7bd, 5ab93287554299753720f78f, 5ae3b4d05542992f92d82349, 5ae77a31554299540e5a55c7, 5ae0d91e55429924de1b7198, 5ae64cab5542991bbc9760be, 5ab865be5542990e739ec8e5, 5a804fc45542992bc0c4a6f0, 5ac2ffa9554299218029dbb2, 5ae7b03e5542993210983ef6, 5a77153355429937353601c8, 5ae61be055429929b0807ace, 5a8ae6c055429950cd6afbce, 5ae64cab5542991bbc9760be, 5a72a00d5542991f9a20c53c, 5ae7b03e5542993210983ef6, 5aeacc75542995085b37473, 5ab9253c554299131ca4227f, 5a8ee0a35542990e94052ba0, 5a866fee5542991e77181657, 5a888a8a5542997e5c09a603, 5abced7e5542993fe9a41da0, 5ae4b3da55429913cc2044d6, 5add28c85542992ae4cec4be, 5abffc8554299012d1db552, 5a8bab4e554299240d9c207c, 5abae52a5542996cc5e49ee, 5abba27f5542996606241708, 5ab6ad2855429953192ad35e, 5aba6b2d55429901930fa7a9, 5abc145b554299658360041f, 5a7336d05542991f9a20c68d, 5ac3b0f15542995ef918c1fc, 5ac3ad225542995ef918c1da, 5a7a06935542990198aef050, 5ae6038155429929b0807a55, 5ab3dde2554299753aec59d6, 5ab381b155429969a97a816b, 5a77bd595542995d83181291, 5a76cb6e5542994aec3b717a, 5a7524ca55429929fddd850a, 5ade025e5542997dc790711e, 5ac17f4f5542994ab5e67d70, 5ae0fa8b5542997b2ef7d0c6, 5a7336d05542991f9a20c68d, 5a8b560855429950cd6afcba, 5adce28f5542990d50227d52, 5ac491eb5542996feb3fe8d2, 5a7fe9975542994857a76847, 5a72b2695542991f9a20c56f, 5a89372855429951533612e6, 5ae219df5542992f1f2b37fc, 5a8a84775542996c9b8d5f19, 5abd7ca05542993062266cab, 5ac07a585542996f0d89cbf0, 5ae8e171b554299068b959e5a, 5a79e0445542994f819ef0e7, 5ae0d26455429945ae959473, 5a8f0e065542997ba9cb319c, 5adce28f5542990d50227d52, 5a8f7de3554299458435d657, 5adc1309554299438c868d3b, 5ac219df5542992f1f2b37fc, 5ae80043055429969796c1ba0, 5ac39f2a554299391541382d, 5a72b1c25542992359bc3172, 5abe3f9455429976d4830aa, 5a8f7de3554299458435d657, 5ab74412554299110f219ae8, 5a904e725542995651fb5118, 5a7a02235542996c55b2dcd3, 5acdc318c5542996e685252d5, 5a78cdf7554299029c4b5e9f, 5ade8f5e55429975fa854f11, 5ab865be5542990e739ec8e5, 5abca9df5542996cc5e49f45, 5adc28c5542994ed6169c30, 5a7e7bf45542994959a199d6, 5adbe1e755429947f1f73853, 5a83168855429966c78a6b2e, 5adc134b5542994650320c5c, 5a90c58255429916514e756c, 5a8efd3c55429918e830d179, 5abbbdc135542993f40c73bf6, 5add7d055542990dbb2f7e61, 5ab344af554299753aec5969, 5a8a35625542992d82986efd, 5ab3dad4554299753aec59cb, 5a8dc8de55429941ae14e060, 5ae377155542991a06ce99c7, 5a7cb48a5542996cd594b9a1, 5ac143535542991316484aac, 5ac31c9d554299741d48a203, 5ae5569255429908b63265e4, 5ab93287554299753720f78f, 5abd04f15542996e802b467e, 5a72b2695542991f9a20c56f, 5ab59b045542997d4ad1f190, 5a7f3d325542992e7d278cb5, 5ae061d5554299603e41840e, 5ae56d31554299546bf82ed7, 5ae255db5542992decdbccc1, 5ab6e856554299710c8d1fac, 5a7a358f5542990783324ec1, 5a7f38ae5542992e7d278c99, 5ab5c9c5554299494045f065, 5ac061ab554299294b218fac, 5a8ee0a35542990e94052ba0, 5ae3bdfa5542990afbd1e1c0, 5ab561d85542992aa134a23c, 5ac3d8dc5542992f92d8239c, 5a7bb3d9554299294a54aaa0, 5abbb1f745542996cc5e49fb5, 5adce28f5542990d50227d52, 5a904e725542995651fb5118, 5add992c5542997545bbdb83, 5adc1309554299438c868d3b, 5adfd35b55429906c02daa54, 5ab39701554299233954ff5e, 5ab58b58955429950cd6afcc2, 5ad22d035542996483e4925, 5a7fa53c5542995d8a8ddedc, 5a84322b5542996488c2e50d, 5ab8d0065542997b22f6a6e9, 5a8d82fc5542997545bbdb57, 5a80d30655429938b61421fe, 5a72b2695542991f9a20c56f, 5a81ff1d554299676cecb1c3, 5a755665542997b22f6a6e9, 5a79e0445542994f819ef0e7, 5ae4c2145542995dadf243e7, 5abbbd0f55429931dba144d5, 5a7cc9c9554299452d57ba72, 5a7bb3d9554299294a54aaa0, 5a835f9554299123d820f3, 5ab5141a5542991779162d70, 5ae4c2145542995dadf243e7, 5ae2e27155429928c423952a, 5abec5e25542994516f45f73, 5ab698885542995eadef002a, 5a7f98e65542996796c1ad8, 5a77bd595542995d83181291, 5a78de46554299148911f9a6, 5ae377155542991a06ce99c7, 5ae6140555429966de7b71b2a, 5a823ae45542990a1d231f6d, 5ab520665542996a3a96a02a, 5ae168865542994ab5c67d14, 5ac1944c5542996f0d89cc90, 5a7cedca55429909bec7689c, 5ab707c05542991d3223760, 5ae27edc5542992decdbcd2d, 5ab979da554299131ca4233a, 5ab345db55429969a7a8122, 5a88fea05542997e5c09a6e9, 5ae38ef5542994611c8b437, 5ab39701554299233954ff5e, 5add992c5542997545bbdb83, 5ab5e6d65542997d4ad1f232, 5a88b7735542993e715ac079, 5adfd35b55429906c02daa54, 5a8514545542992a431d1ad2, 5adfd35b55429906c02daa54, 5ae538ef5542994611c8b437, 5ab52065542996a3a96a02a, 5a74fbc55542996c70cfac63, 5ab55435554299488d4d9939, 5ae31a9c55429928c42395ef, 5ab67b8f55429954757d32f0, 5ae13f525542997b2ef7d169, 5a7d1f605542995ed0d165fb, 5ade52e85542997c77adedfa, 5a7607d7554299109176e61a, 5a85603a5542997b5ce3fff1, 5ac17f4f5542994ab5c67d70, 5a7fa53c5542995d8a8ddedc, 5abaef34554299660624169c, 5ae3d8dc5542992f92d8239c, 5ae0fa8b5542997b2ef7d0c6, 5ab55435554299488d4d9939, 5a904e725542995651fb5118, 5a879c8e5542994846c1cd3b, 5a870d05542996309b710, 5ab3dde2554299753aec59d6, 5ac3ad225542995ef918c1da, 5ae11a6755429901ffe4ad8d, 5ab9116f5542991b5579f0db, 5ae755665542997b22f6a6e9, 5ae316f355429928c42395e3, 5abfb45542997ec76fd440, 5a88377c5542997e5c09a5a7, 5a8099025542996402f6a588, 5a74248855429929fddd83e5, 5ac39a1c554299657fa290f9, 5abbc70d55429922cd8e7f9b, 5ae13f525542997b2ef7d169, 5ac1f7f355429964131be2ae, 5a84322b5542996488c2e50d, 5a7738dc554299373536021f, 5a760f6855429976ec32bcf9, 5a7f38ae5542992e7d278c99, 5ae655c855429908198fa599, 5a821ffa5542990a1d231f5c, 5a90c2b35542995651fb51df, 5a78ed46554299148911f9a6, 5a8454e85542992ef85e23be, 5a8514545542992a431d1ad2, 5ac168865542994ab5c67d14, 5a88b7735542993e715ac079, 5a77aff5542992a6e59df86, 5ab39701554299233954ff5e, 5ac219df5542992f1f2b37fc, 5ab67b8f55429954757d32f0, 5a7a0d455542990783324e13, 5a8461d55542990548d0b29b, 5a879ab055429964ef30887e, 5ae5365d5542992663a4f16d, 5a7a0d455542990783324e13, 5a7f99e855429969796c1af3, 5ae5365d5542992663a4f16d, 5a736bfa5542991f29ee2e03, 5abfb45542997ec76fd440, 5a8f8f345542997ba9cb32c2, 5ab912155429919ba4e238a, 5a8dfbfb5542995085b3736e, 5a8a35625542992d82986efd, 5ac31c9d554299741d48a203, 5ae5365d5542992663a4f16d, 5add28065542990d50227e08, 5ae64cbf5542992ae0d162c1, 5adc134b5542994650320c5c, 5ac31c9d554299741d48a203, 5adf2b325542993a75d2640b, 5ae755665542997b22f6a6e9, 5a8454e85542992ef85e23be, 5a7cc5ae55429909bec767fc, 5a8a84775542996c9b8d5f19, 5ae377a35542994393b9e6db, 5ac4fa8c5542992a4173fb536, 5a77aff5542992a6e59df86, 5ac31a9c55429928c42395ef, 5adf5ebd5542995ec70e8fd8, 5a8a4bdc55429930ff3c0d8c, 5ae77a31554299540e5a55c7, 5ac2adf3554299657fa2900f, 5ab5a2f85542997d4ad1f197, 5abd7cb855429924427fd00a, 5ae136f655429920d5234325, 5ae525835542990ba0bbb1cd, 5a7738dc554299373536021f, 5a7a52745542996c55b2dd4f, 5ae1f61a5542994d89d5b2e1, 5add28c85542992ae4cec4be, 5a8bdef85542997f31a41dea, 5ae614055542996de7b71631b, 5a7336d05542991f9a20c68d, 5aeacc75542995085b37473, 5a8cdc5255429941ae14df21, 5ae664955542992aead01631b, 5ae2aba15542996483e64a32, 5abba27f5542996606241708, 5abd7ca05542993062266cab, 5ac1a5cd55429947d6ccce94, 5a736bfa5542991f29ee2e03, 5a8f0e065542997ba9cb319c, 5a8a2d805542996c9b8d5e2e, 5ae546e85542992663a4f1b5, 5ab6e856554299710c8d1fac, 5aba0e675542994dbf0198a0, 5ae3345f55429928c4239682, 5a7a02235542996c55b2dcd3, 5ac4fa8c5542992a4173fb536, 5a8beddd5542995d1e6f1468, 5abd90545542996e802b47d7, 5a7e39515542995ed0d166da

MuSiQue Example Ids

```

2hop_ 376129 44537, 2hop_ 764465 126539, 3hop1_ 434518 136629 55288, 2hop_ 353084 36340,
2hop_ 344450 160798, 2hop_ 637856 351187, 2hop_ 760990 44191, 3hop1_ 162325 11248 3752,
2hop_ 326799 278127, 2hop_ 239927 62031, 2hop_ 153813 69936, 3hop1_ 213491 782843 75255,
2hop_ 2846 2741, 2hop_ 3880 909, 2hop_ 347735 36735, 2hop_ 144393 87372,
4hop1_ 709382 146811 31223 45305, 2hop_ 143434 20122, 2hop_ 21457 74218,
3hop1_ 129597 517267 451901, 2hop_ 469317 776926, 2hop_ 27032 5400, 3hop2_ 83954 32417 24628,
3hop2_ 14790 57411 86234, 2hop_ 78490 49700, 3hop1_ 228008 354329 5303, 2hop_ 631861 160851,
3hop1_ 662283 507729 351187, 2hop_ 482727 20661, 3hop1_ 858308 102146 84004, 2hop_ 565717 77346,
3hop1_ 470555 668347 492654, 2hop_ 25478 65517, 2hop_ 129389 31248, 2hop_ 527889 5365,
2hop_ 20857 20779, 2hop_ 770 919, 2hop_ 375649 80178, 3hop1_ 332614 131794 17114,
2hop_ 144295 211364, 2hop_ 108160 159045, 2hop_ 46545 88521, 2hop_ 518906 44191,
2hop_ 733628 131886, 4hop1_ 28235 74795 84660 15312, 2hop_ 104341 92821, 2hop_ 445544 127008,
2hop_ 46766 79233, 2hop_ 342213 185893, 2hop_ 528837 126102, 2hop_ 497897 541630,
3hop1_ 48619 26424 581618, 2hop_ 87287 83906, 4hop1_ 411538 805015 475503 32631,
2hop_ 658198 72962, 2hop_ 42307 120207, 2hop_ 30878 555599, 3hop1_ 8373 87072 45358,
3hop2_ 337255 48727 83343, 2hop_ 251450 8796, 3hop1_ 161080 639509 644660, 2hop_ 568231 52667,
2hop_ 424189 49441, 3hop1_ 821692 74047 756423, 2hop_ 531731 79705, 3hop1_ 257981 259472 611044,
2hop_ 370765 14904, 2hop_ 446352 14183, 2hop_ 81087 13292, 2hop_ 684971 333904,
2hop_ 234176 69926, 2hop_ 858097 121880, 4hop2_ 724536 444580 75897, 631997, 2hop_ 492509 70585,
4hop1_ 405751 4520 65397 49736, 2hop_ 128610 126060, 3hop1_ 325154 786384 42990,
2hop_ 34130 56335, 2hop_ 145997 63766, 2hop_ 146446 690423, 2hop_ 225632 11125, 2hop_ 856457 495,
2hop_ 129234 330515, 2hop_ 15674 42467, 3hop1_ 161946 84298 53741, 2hop_ 48959 83539,
2hop_ 64650 20556, 3hop1_ 316518 395352 131877, 2hop_ 136618 92216, 2hop_ 199336 185893,
2hop_ 930 57555, 3hop1_ 31942 48661 15069, 2hop_ 35105 160978, 2hop_ 128804 351187,
2hop_ 153004 86587, 2hop_ 715365 565667, 2hop_ 401484 135138, 2hop_ 52622 67783,
2hop_ 713501 58946, 2hop_ 300786 39199, 2hop_ 5430 5348, 3hop2_ 29467 132027 73594,
3hop1_ 225298 755188 480696, 2hop_ 367037 80178, 2hop_ 343473 53204, 2hop_ 848923 66214,
3hop1_ 369072 287321 161879, 2hop_ 250315 64214, 3hop1_ 104311 833580 61459, 2hop_ 1835 322987,
3hop1_ 836616 291186 4303, 2hop_ 531924 1094, 2hop_ 131831 84128, 2hop_ 328708 90697,
2hop_ 704691 82816, 2hop_ 80353 3001, 2hop_ 196785 61424, 2hop_ 130964 47336,
3hop1_ 761109 548045 159613, 3hop1_ 4525 52205 55099, 3hop1_ 58522 787757 69397,
2hop_ 58284 37793, 2hop_ 487591 7672, 2hop_ 250913 58115, 2hop_ 31095 85298,
2hop_ 144937 8600, 3hop2_ 625639 25582 21116, 3hop2_ 30023 63595 53125, 2hop_ 584872 88978,
2hop_ 116643 351162, 2hop_ 826203 62031, 2hop_ 85036 909, 2hop_ 62996 299942,
2hop_ 236731 229413, 2hop_ 15169 87091, 2hop_ 143791 75878, 2hop_ 658198 90536,
2hop_ 70321 15755, 2hop_ 131105 68117, 2hop_ 143162 438686, 2hop_ 20771 65517,
2hop_ 65149 46180, 2hop_ 251426 88653, 3hop1_ 238983 403313 61770, 2hop_ 28291 709757,
2hop_ 391909 3430, 3hop1_ 266733 291186 50964, 2hop_ 205685 160137, 2hop_ 343141 702969,
3hop1_ 383692 434040 59381, 2hop_ 240975 736878, 2hop_ 507864 368521, 3hop1_ 723003 593059 76293,
2hop_ 109234 62766, 4hop1_ 16401 4520 65397 52251, 2hop_ 140591 256194,
2hop_ 104757 74309, 2hop_ 194976 55566, 2hop_ 361127 140822, 3hop1_ 108774 104782 14771,
2hop_ 393686 620110 61746 261712, 2hop_ 324178 83854, 3hop1_ 849536 301867 127418,
2hop_ 24408 541630, 2hop_ 54755 729624, 2hop_ 693650 61232, 3hop1_ 89787 49283 632017,
4hop1_ 104663 221169 833580 61459, 2hop_ 664573 36741, 3hop1_ 702271 823374 26254,
2hop_ 129892 62851, 3hop1_ 659125 39490 23352, 2hop_ 222162 386543, 2hop_ 446009 412262,
2hop_ 781841 77980, 3hop1_ 706183 20196 10585, 2hop_ 809948 162428, 3hop1_ 458602 681261 369731,
2hop_ 529082 114112, 3hop1_ 388966 508834 145463, 2hop_ 582169 370960, 2hop_ 225632 52135,
2hop_ 302491 81463, 2hop_ 136889 52356, 2hop_ 81363 42667, 3hop1_ 599980 544161 92922,
2hop_ 504710 513189, 2hop_ 145939 11443, 2hop_ 320353 4018, 2hop_ 27033 85063,
2hop_ 145110 861627, 2hop_ 149891 44359, 2hop_ 376266 37939, 3hop2_ 10879 37094 161133,
3hop2_ 159915 8509 19700, 4hop1_ 15118 31258 43153 32993, 3hop1_ 522518 132413 16066,
2hop_ 129782 517267, 3hop1_ 252998 715836 26008, 4hop1_ 205937 144938 83779 44678,
2hop_ 131318 47465, 2hop_ 338405 68172, 4hop3_ 3153 3356 11988 24628, 2hop_ 106465 54210,
2hop_ 397761 404718, 4hop1_ 632232 164954 6975 6891, 2hop_ 121872 708662, 2hop_ 73501 31113,
2hop_ 378511 191233, 3hop1_ 85045 96305 25007, 3hop1_ 755950 592709 78102,
2hop_ 811421 377891, 3hop2_ 63595 391767 53125, 2hop_ 131380 84859, 3hop1_ 158678 48408 37793,
3hop1_ 7312 830682 68600, 2hop_ 207212 21032, 3hop1_ 10725 695397 74345, 2hop_ 445228 774871,
4hop1_ 603090 818753 783943 26110, 2hop_ 177131 646483, 3hop1_ 801682 192919 16121,
2hop_ 243908 500443, 3hop2_ 89818 157704 4107, 2hop_ 160546 26427, 2hop_ 128772 745471,
2hop_ 62588 20779, 2hop_ 661636 82027, 2hop_ 105388 89066, 2hop_ 368185 131944,
3hop1_ 153577 411195 8682, 2hop_ 327451 90697, 2hop_ 647590 134798, 3hop2_ 30796 804098 24137,
2hop_ 146227 42328, 2hop_ 152881 620955, 2hop_ 11693 42892, 2hop_ 753498 7606,
2hop_ 2795 2741, 3hop1_ 373317 533132 1660, 2hop_ 229374 333904, 3hop1_ 370820 301867 127418,
3hop1_ 713250 4016 83854, 2hop_ 130414 68117, 4hop1_ 7312 84360 334118 41330, 2hop_ 65149 68376,
2hop_ 182310 565529, 3hop1_ 136299 84467 89676, 2hop_ 454055 86874, 2hop_ 604878 40786,
2hop_ 307569 51671, 2hop_ 854082 159115, 2hop_ 198557 55566, 3hop1_ 352446 506157 44678,
2hop_ 468848 44537, 2hop_ 207571 126101, 4hop2_ 53235 18485 57802 311656, 2hop_ 451164 140822,
3hop1_ 37692 84298 53741, 3hop1_ 672119 196807 760519, 3hop2_ 131210 661360 54023,
2hop_ 8531 24846, 3hop2_ 77886 64137 69951, 2hop_ 730762 8600, 2hop_ 350323 45731,
2hop_ 131117 53519, 3hop1_ 157534 275705 81669, 2hop_ 185628 677577, 2hop_ 77119 20732,
2hop_ 67755 82010, 3hop1_ 790278 593059 76293, 3hop2_ 162189 611045 73761, 2hop_ 568848 50788,
2hop_ 45625 61952, 2hop_ 146207 30651, 2hop_ 57439 78714, 2hop_ 3756 52135,
3hop1_ 501828 348668 856982, 3hop1_ 106423 35178 686699, 2hop_ 103203 23140,
3hop1_ 77985 66386 16350, 2hop_ 664921 579740, 2hop_ 106125 20644, 2hop_ 400998 61424,
3hop1_ 35884 161545 16532, 2hop_ 584521 755188, 2hop_ 80508 400874, 2hop_ 664137 58115,
2hop_ 453207 80674, 3hop1_ 29335 30907 24600, 2hop_ 144364 68900, 2hop_ 226817 482901,
4hop3_ 39198 75897 8509 19700, 2hop_ 713863 64008, 2hop_ 71269 36735, 2hop_ 504228 64689,
2hop_ 604878 18657, 2hop_ 81372 303417, 3hop1_ 674688 707133 72062, 2hop_ 157766 18657

```

Chapter 6

Benchmarking Multi-Step Reasoning and Tool Use

6.1 Introduction

In Chapter 5, I introduced an RL agent capable of performing sequential decision-making in the domain of question-answering and mathematical reasoning. In this chapter, we propose COMPASS-QA (Challenging Open-ended Multi-step Problems And Sequential Search), a new benchmark to measure the ability of current models to perform multi-step reasoning and tool use, and to train more advanced LLM-based RL agents. Here, we provide 700 challenging questions that require multiple steps of reasoning and tool use to effectively answer. Additionally, we provide trajectories generated by human annotators outlining the steps they took to reach the final answer. We hope that this dataset will help researchers meaningfully benchmark the effectiveness of new models on multi-step reasoning and tool use.

Large language models (LLMs) have demonstrated incredible capabilities in information processing and solving complex tasks [69, 12, 162]. However, as discussed in Chapter 5, many high-value tasks require information or capabilities that are beyond the model’s internal knowledge base. For example, answering a question about recent events may need access to live internet search, solving a multi-step math problem might require using a calculator or theorem-proving tool, and completing a complex engineering task could require executing code, writing unit tests, or reading documentation about a software framework. The ability to seamlessly identify the need for a tool call, generate a well-formed query to invoke that tool, and incorporate the result of executing that query to plan subsequent steps is key to solving many real-world challenges. Therefore, this motivates the need for new benchmarks that can capture and evaluate such capabilities.

In this chapter, we introduce a benchmark that targets multi-step reasoning and tool use

capabilities, with a focus on complex knowledge-intensive tasks. Our benchmark focuses on two of the most powerful and ubiquitous tools for information access: search and web browsing. Our dataset consists of 700 multi-step knowledge-intensive questions. In addition to questions and solutions, we also provide multi-step human-generated reasoning, search, and browsing trajectories for each query. This dataset can be used for both training and evaluation purposes.

In order to solve these tasks, AI systems must go beyond simple information retrieval and search. They must have the ability to interact with external knowledge resources, iteratively use search and browsing tools, and perform multi-step reasoning to intelligently orchestrate a sequence of such actions. Examples of such problems include diagnosing a software bug by searching for error codes and reading documentation, or answering historical questions that require synthesis of information from multiple primary sources. Furthermore, these tasks often require a chain of reasoning where the output of one step (e.g., a search result) informs the input and the action for the next step (e.g., browse a specific webpage identified in the top search results).

Traditional question-answering datasets (e.g., SQuAD, Natural Questions, TriviaQA) are single-step and can be solved with a simple search query or retrieval from the provided documents. Other benchmarks, such as HotPotQA, Musique, BeerQA and CofCA which I evaluated on in Chapter 5, require multi-hop retrieval from given datasets, but do not involve external tool interactions. Recent open question-answering benchmarks such as SimpleQA are not designed for evaluating multi-step capabilities. Recent agentic environments (e.g., GAIA, AgentBench) explore complex environment interactions with several tools for real or simulated web environments. While valuable, these settings can introduce additional complexity, such as the ability to perform multi-modal reasoning and visual processing, meaning that these benchmarks cannot be used to evaluate the multi-step reasoning and tool use capabilities of pure language models. To our knowledge, this benchmark is the only dataset with accompanying reasoning and tool use traces generated by human annotators.

Our core contributions are as follows:

- We compiled a challenging dataset of questions that requires multiple steps of search and web browsing.
- We provide human annotations capturing the exact sequence of actions necessary to answer these challenging queries.
- We benchmark top-performing open-source (Gemma 3, Llama 4, and DeepSeek R1) and proprietary models (Gemini 2.5 Pro, Claude 3.7 Opus, and ChatGPT 4.5) against this new dataset.

6.2 Data Collection

We worked with human annotators at Surge AI to capture the sequence of steps necessary to answer a challenging, knowledge-intensive question.

6.2.1 Sourcing and Generating Queries

The root of the dataset was the challenging multi-hop questions, so we will start by describing how we sourced, generated, rewrote, and filtered these queries. These questions were compiled by filtering Gemini logs for questions that received thumbs-down feedback from users and which Gemini classified as ones that would benefit from multiple steps of search and web browsing. These questions were then rewritten and sanitized by Surge crowdworkers.

6.2.2 Filtering and Rewriting Queries

In this task, the human annotator filtered and rewrote questions from the previous stage. The purpose is to curate a set of challenging, high-quality prompts that require (or would at least benefit from) multi-turn search and web browsing. These will be used as input to the next phase.

We input 3000 questions into this data filtering and rewrite pipeline, and received 700 rewritten questions that were high-quality and multi-step as judged by the human annotators, i.e., 24.3% were preserved after this filtering step.

6.2.3 Generating Trajectories with Human Annotators

To generate trajectories for each question, we provided the human annotator with access to a search engine and web browser, and allowed them to take any number of search and web browsing actions before answering the original question.

Concretely, at each time step, the human annotator can take one of six actions:

- **SEARCH:** Issue a Google search query
- **SAVE_TEXT:** Save text, e.g., a search snippet or a paragraph on a webpage.
- **SAVE_IMAGE:** Save an image, e.g., a figure or relevant photo.
- **CLICK:** Open a URL, e.g., one of the search results or a link in a webpage.
- **CTRL-F:** Perform a keyword search within a webpage.
- **ANSWER:** Synthesize an answer to the original user question.

The trajectory terminates when the human annotator chooses the “ANSWER” action.

Example Trajectory

To show the type of trajectories collected, we provide an example below. Note that we picked a relatively simple example for illustrative purposes.

Question: “Is globbing done before or after shell expansions?”

Given the above question, we would like to collect the following information:

Step 0: User searches “What is globbing?”

Here is the data that should be collected for this step:

‘action’: SEARCH, ‘thought’: “First, I want to make sure I understand what globbing in bash means, so I search for “What is globbing in bash?”, ‘input’: “What is globbing?”, ‘output’: <SEARCH_RESULTS>

<SEARCH_RESULTS> is a list of 5 search results in the following format:

[index] <source_title>\n\n<snippet>\n\n<url>

For example, the first search result should be stored as:

[1] What is Globbing? - Definition from Techopedia

Globbering is the process of using wildcard characters to request or evaluate sets of files with the same partial names or sets of characters.

<https://www.techopedia.com> › definition › globbing

Step 1: Now the user reads a snippet.

‘action’: READ, ‘thought’: “The first snippet seemed relevant so I read it.”, ‘input’: “<TEXT>”, ‘output’: None <TEXT> is the following snippet and URL stored as a string. Here we assume that the user read URL 1.1 from Step 0.

Step 2: Next the user generates a new query which is “What are shell expansions?” and runs a Google search.

‘action’: SEARCH, ‘thought’: “I was satisfied with the first snippet and feel that I understand what globbing is, so I decide to make another search to learn more about shell expansions.”, ‘input’: “What are shell expansions?”, ‘output’: <SEARCH_RESULTS>

<SEARCH_RESULTS> is the following (stored as a string)

Step 3: The user then reads a snippet.

‘action’: READ, ‘thought’: “The first snippet seemed helpful because it lists the kinds of expansions, so I read it.”, ‘input’: “<TEXT>”, ‘output’: None <TEXT> is the following snippet and URL stored as a string. Here we assume that the user read URL 2.1 from Step 2.

Step 4: The user clicks on the first link from Step 3 and transitions to a new webpage:

‘action’: CLICK, ‘thought’: “The first snippet looked helpful, so I clicked on the first result <URL2.1>”, ‘input’: “<URL2.1>”, ‘output’: <WEBPAGE_HTML> <URL2.1> is the webpage corresponding to the link the user clicks on. We assume the user clicked on URL2.1. <WEBPAGE_HTML> is the content of the webpage corresponding to URL2.1 (shown below) stored as a

string. .

Step 5: Next the user runs a Ctrl + F for keyword: “order”

‘action’: CTRL_F, ‘thought’: “I want to see if the page contains useful information about the order of shell expansions, so I control-f for “order”.”, ‘input’: “order”, ‘output’: <KEYWORD_RESULTS>

<KEYWORD_RESULTS> is a list of indices in the webpage that represent occurrences of the keyword.

Step 6: Read Text

‘action’: READ, ‘thought’: “The first keyword search result looks useful, so I read the paragraph containing it.” ‘input’: “The order of expansions is: brace expansion; tilde expansion, parameter and variable expansion, arithmetic expansion, and command substitution (done in a left-to-right fashion); word splitting; and filename expansion” ‘output’: None

Step 7: User gives final answer

‘action’: ANSWER, ‘thought’: “I now have all the information I need to reply. From my first search “What is globbing?” I know that globbing is filename expansion, and from the text above [1.1]. Therefore, globbing is done after all other shell expansions.” ‘input’: ‘Globbing is done after all other shell expansions. To be more specific, there are 7 types of shell expansions, and they are executed in the following order: brace expansion; tilde expansion, parameter and variable expansion, arithmetic expansion, and command substitution (done in a left-to-right fashion); word splitting; and filename expansion (aka globbing).’ ‘output’: None

6.3 Evaluation Benchmark

In this section, we will describe the procedure to evaluate a model with this dataset. This benchmark relies upon the use of an LLM-as-a-judge in order to accurately evaluate performance.

6.3.1 LLM as a Judge

We adopt model-based evaluation, which is emerging as a strategy that combines the accuracy of human evaluation with the cost-effectiveness and low latency of automated metrics [238, 78].

Because the questions are relatively complex, evaluating the quality and correctness of the answers requires a model with strong reading comprehension. In this chapter, we use Gemini 2.5 Pro as our model judge, as we found that it meets our need for a highly discerning, low-latency model.

To assess the answer to each question, we prompted Gemini 2.5 Pro to judge the correctness of each answer compared to a golden answer (the human annotator’s final response).

6.3.2 Data Format

Below, we describe the JSON format in which we would like you to log the sequence of actions. Each action has a slightly different set of expected inputs and outputs, so we will provide a high-level template for each action type as well as a concrete example that includes at least one instance of each action.

```
Action = Enum('Action', ['SEARCH', 'CLICK', 'READ', 'CTRL_F'])
'action': SEARCH, 'thought': <THOUGHT>, 'input': "<QUERY>",
'output': <SEARCH_RESULTS>, 'state': <CURRENT_URL> 'action': CLICK, 'thought':
<THOUGHT>, 'input': "<URL>", 'output': <WEBPAGE_HTML>, 'state': <CURRENT_URL>
'action': READ, 'thought': <THOUGHT>, 'input': "<TEXT>", 'output': None, 'state': <CUR-
RENT_URL> 'action': CTRL_F, 'thought': <THOUGHT>, 'input': "<KEYWORD>", 'output':
keyword_result=<KEYWORD_RESULT>, progress=X/Y, 'state': <CURRENT_URL>
```

where X is the index of the keyword result and Y is the total number of keyword matches in the document.

Note that `<WEBPAGE_HTML> = fetch_html(<URL>, date)` if date is supported.

For more example trajectories, see Appendix 6.H.

6.3.3 Dataset Composition

As shown in Table 6.3.1, we divided the dataset into 279 train examples, 100 dev examples, and 300 test examples. This provides enough examples to meaningfully develop and evaluate a new methodology, while also offering the opportunity to train on a reasonably large number of human-annotated trajectories.

Data Split	Dataset Size
Train	279
Dev	100
Test	300

Table 6.3.1: Number of examples in each split of the dataset.

6.4 Dataset Validation and Analysis

In this section, we describe how the dataset was cleaned, formatted, and validated, and provide insights into the composition of the dataset.

6.4.1 Dataset Validation

The COMPASS-QA dataset underwent a meticulous multi-stage validation procedure to ensure the integrity, safety, and utility of its contents. This process combined human oversight with large language model capabilities for scale and consistency.

- **Stage 1: Initial Manual Question Filtering.** All candidate questions (approximately 700) were first subjected to a manual review by the research team. This step aimed to filter out questions that were unclear, trivial, inappropriate, or otherwise unsuitable for the benchmark’s focus on multi-step reasoning, reducing the set to roughly 600 questions.
- **Stage 2: Automated Content Moderation.** To systematically screen for any potentially inappropriate or sensitive content within the detailed human-annotated trajectories, Gemini 2.5 Pro was utilized. The model was tasked with reviewing every textual element: the questions, the annotators’ thought processes, the actions logged, and the textual results of those actions. This automated moderation flagged approximately 200 utterances across the dataset.
- **Stage 3: Manual Review of Flagged Content.** The 200 utterances identified by Gemini 2.5 Pro were then individually reviewed by human evaluators. This critical human-in-the-loop step confirmed the automated flags and led to the removal of a further six questions, ensuring the dataset met appropriate content standards.
- **Stage 4: Property Tagging.** To enhance the dataset’s utility for researchers, Gemini 2.5 Pro was employed to automatically tag each question with several important characteristics. These tags included assessments of the question’s open-endedness, its time-sensitivity, and whether it required multi-modal understanding, providing valuable metadata for future analyses and model evaluations.

6.4.2 Multi-Hop Nature

In Figure 6.4.1, we provide statistics for the number of hops across all human trajectories in our dataset. Interestingly, we see no examples of 2-hop trajectories, suggesting that we have successfully filtered out questions that require only one step of search or web browsing. Recall that the act of answering the original question counts as an action, so any trajectory that involves calling a tool more than once must contain at least three steps. However, there is one trajectory consisting of a single action, meaning that the human felt confident to answer the question without performing any search or web browsing. Although this single trajectory should have been filtered out, the overall statistics are encouraging and suggest that this dataset consists of challenging multi-hop questions.

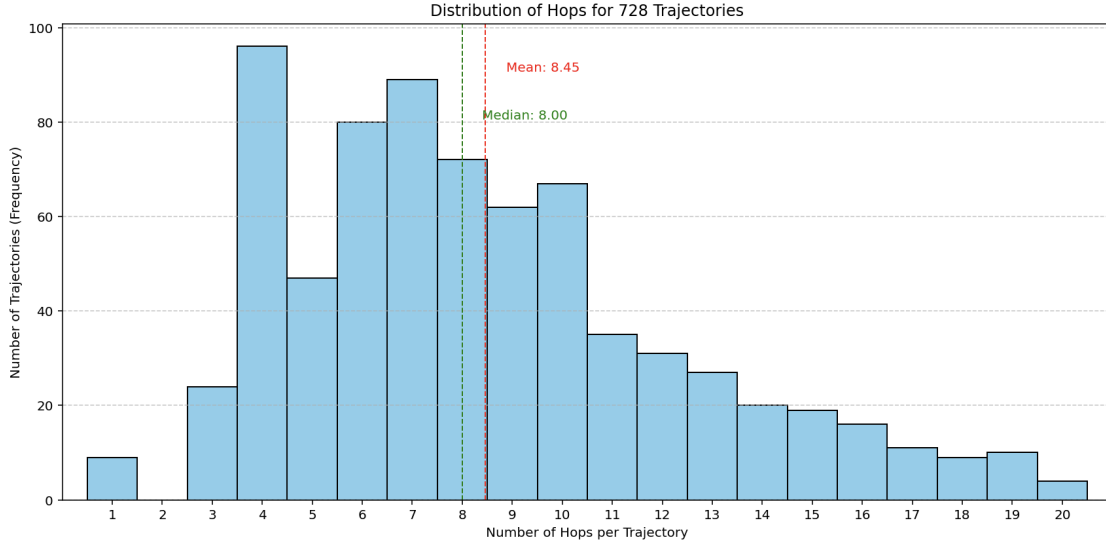


Figure 6.4.1: Histogram depicting the number of hops across all trajectories.

6.4.3 Distribution Over Actions

In Figure 6.4.2, we depict the distribution over actions found in these trajectories. As can be seen from this figure, the most common actions were searching, clicking URLs, and saving text, with each comprising about 25% of all actions. Searching within webpages (Ctrl-F) was a relatively rare action at 6.4%. All episodes contain exactly one synthesize step to generate a final answer to the original question.

6.5 Rubric-Based LLM Evaluation

As the field moves toward more challenging agentic tasks, LLMs must be capable of providing high-quality responses to open-ended questions. However, in order to achieve these capabilities, we must be able to effectively measure such capabilities. Prior datasets like HotPotQA [226], BeerQA [167], MuSiQue [203], and CofCA [222] have a single correct answer (or set of alias answers), and cannot measure the ability of a model to answer challenging open-ended questions. To address this gap, the natural next step is to employ LLMs to perform synthetic evaluations that mimic those of human annotators, but at much lower cost and latency. Here, we propose to augment LLM judges with LLM-generated rubrics, mimicking a best practice used to improve the consistency and inter-rater agreement of human annotators. To further adopt best practices inspired by human evaluation, we also report scores on a 5-point Likert scale from “Poor” to “Excellent”, offering richer signal on the quality of a model response compared to a binary correctness label.

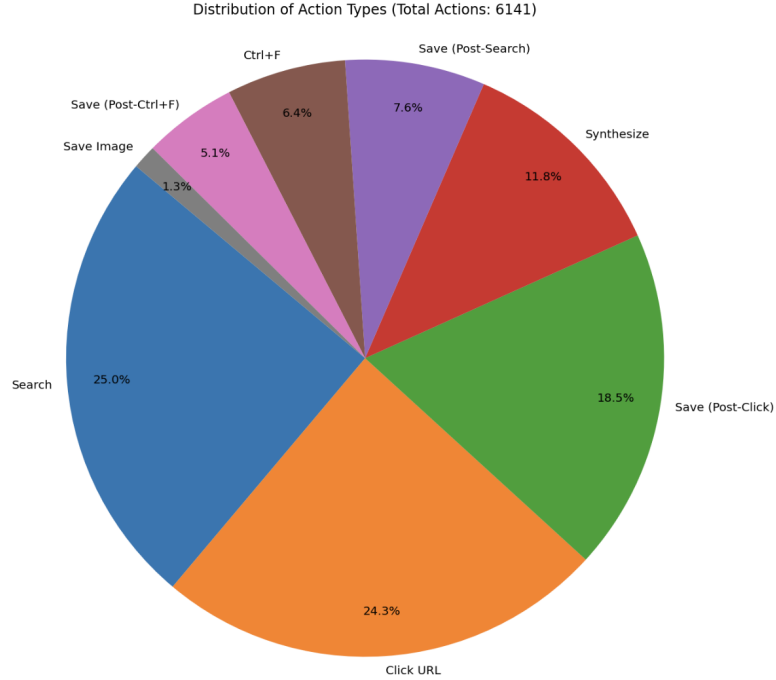


Figure 6.4.2: Piechart depicting the composition of trajectories with respect to action types.

6.6 Experiments

In this section, we evaluate open-source and proprietary models using the rubric-based LLM framework described above, yielding the results in Table 6.6.1. This is of course due in part to the nature of this task, where we only consider the answer to be correct if it is judged to be “Excellent” on a 5-point Likert scale. Overall, the LLM judges were remarkably consistent with each other, which was encouraging. In terms of performance on the task itself, o3-mini performed best (10.7%), with Gemini 2.5 Pro as a close second (9.9%). Claude 3.7 Sonnet did surprisingly poorly (7.2%), underperforming even the open-source Gemma 3 27b model (8.2%). The most striking aspect of these results was how poorly current frontier models performed with scores hovering around 10%. These results indicate that this benchmark could potentially be useful for hill-climbing and improving the capabilities of frontier models.

6.7 Related Work

Knowledge-based question-answering is a key application of language models and several important benchmarks have targeted this task, including SQUAD [170], TriviaQA [95], Natural Questions [119], FELM [35]. As model capabilities have advanced, many of these earlier benchmarks have become saturated. More recent benchmarks such as SimpleQA [218] focus on queries with concise and short answers, whereas LongFact [219] contains open-ended questions with longer answers.

Grading Model Generating Model	Claude 3.7 Sonnet	Gemini 2.5 Pro	Gemma 3	o3-mini	Average
Claude 3.7 Sonnet	0.070	0.143	0.027	0.047	0.072
Gemini 2.5 Pro	0.093	0.110	0.107	0.087	0.099
Gemma 3	0.077	0.053	0.107	0.090	0.082
o3-mini	0.150	0.167	0.047	0.063	0.107

Table 6.6.1: Percentage of ‘Excellent’ Grades out of 300 example answers graded on a 5-point Likert scale, where ‘Excellent’ corresponds to the highest possible score.

Generating Model	Grading Model	Excellent	Very Good	Good	Fair	Poor
Claude 3.7 Sonnet	Claude 3.7 Sonnet	21	53	87	101	38
	Gemma 3	8	64	166	58	4
	Gemini 2.5 Pro	43	60	89	71	37
	o3-mini	14	71	73	90	52
Gemini 2.5 Pro	Claude 3.7 Sonnet	28	38	83	100	51
	Gemma 3	32	57	139	62	10
	Gemini 2.5 Pro	33	56	77	82	52
	o3-mini	26	69	70	97	38
Gemma 3	Claude 3.7 Sonnet	23	34	66	119	58
	Gemma 3	32	25	158	67	18
	Gemini 2.5 Pro	16	32	83	110	59
	o3-mini	27	31	54	117	71
o3-mini	Claude 3.7 Sonnet	45	50	77	92	36
	Gemma 3	14	71	167	44	4
	Gemini 2.5 Pro	50	75	80	74	21
	o3-mini	19	59	89	89	44

Table 6.6.2: Full Distribution of LLM-Generated Grades over 300 example answers and reported on a 5 point Likert scale.

FreshQA focuses on open-ended queries and evaluates model’s ability to answer questions with misleading premises or queries with time-sensitive answers. TruthfulQA [129], HalluQA [41], CREPE, [230], and QA^2 [115] evaluate model’s ability to factually answer questions in the presence of misconceptions or counterfactual premises. Other datasets, such as TimeSensitiveQA[38], SituatedQA[236], StreamingQA[134], RealTime QA[109], FreshLLMs [212] measure model’s ability to answer questions whose answers change over time.

The most relevant prior work are multi-hop question-answering datasets [226, 167, 84, 222, 203, 72, 60, 197]. Answering these questions requires combining information from multiple sources. HotpotQA [226], which is built on Wikipedia, requires the models to retrieve information across multiple documents and synthesize a final answer based on that. The model is also asked to provide supporting evidence. 2WikiMultiHopQA [84] is similar to HotpotQA, but relies on manual compositional rules. IIRC [60] is also based on English Wikipedia and the accompanying paragraphs offer only partial answers to the questions, and the model is required to locate the missing information by following the links and acquiring and processing knowledge contained in those links. MuSiQue [203] targets questions that would require the model to generate a sequence of 2-4 reasoning steps, while StrategyQA [72] evaluates models’ implicit, multi-step reasoning ability with final answers in yes/no format. ComplexWebQuestions [197] provides questions that require combining information from multiple web snippets or performing multi-step reasoning. These benchmarks evaluate models’ ability to retrieve, sequentially reason, and integrate scattered information to answer complex queries.

While these datasets are all valuable, a differentiating factor for our benchmark is that we provide both a large collection of questions that require multi-step search and browsing, and release trajectories generated by expert human annotators, with clear tags for each action such as generating new queries for search or navigating the web to browse new websites. These trajectories could be of high value for both training the model to navigate multi-step processes and for evaluating models’ process-based and outcome-based performance.

6.8 Conclusion

This dataset is designed to benchmark multi-step tool use, and contains challenging question-answering tasks that require search and web navigation. We also provide human annotations to enable deeper analysis, including training based on process or outcome-based rewards. We hope this enables better benchmarking of models’ ability to perform knowledge-intensive tasks. In particular, in the future, we’d like to use it to extend our work on SWiRL from Chapter 5.

Appendix

6.A Filter and Rewrite Input Queries for Multi-Hop Search

Below, we include the instructions that were shared with human annotators to complete the task of filtering and rewriting input queries.

6.B Summary

In this task, the human annotator will filter or rewrite input questions. The purpose is to curate a set of challenging, high-quality prompts that require (or would at least benefit from) multi-turn search and web browsing. These will be used as input to [Surge Instructions] Multi-Hop Search & Response Generation for Complex Questions.

We have gathered a set of 3k prompts for use as input to this query filter / revision pipeline (we expect 1/4 to 1/5 to be useful, each prompt begins with <PROMPT_START> and ends with <PROMPT_END>):

6.C Instructions

Given a particular query, please indicate whether it should be:

- ACCEPTED
- REWRITTEN (include rewritten text)
- REJECTED (include reason for rejection)

6.D Example 1: Accept Query

If the query is well-written, standalone, and would clearly benefit from one or more calls to a search engine, then please indicate acceptance of this query. Here is an example of a query that should be accepted:

Query: What permission is needed to export saved search results to CVS or Excel?

Category: ACCEPT

Query: Common loss functions and evaluation in segmentation

Category: ACCEPT

Output: What are common loss functions and evaluation methods in image segmentation?

6.E Example 2: Rewrite Query

If a query would benefit from one or more calls to a search engine, but is poorly written or under-specified, please rewrite the query to make it a prompt that you would accept “as-is”. Please also rewrite queries to be clear questions. Here is an example of a query that should be rewritten (and a possible rewrite for it):

Query: detailed tutorial on javascript strict mode

Category: REWRITE

Output: write detailed tutorial on javascript strict mode

6.F Example 3: Query Rejected

If the query is incoherent or wouldn’t benefit from calling Google Search (or could be trivially answered just from the search snippet), please reject it. Here is an example of a query that should be rejected:

Query: Please solve this equation: $48(5(43(\$500+7\%))) = ?$

Category: REJECT

Reason: “No need to search”

Query: Write a seo friendly youtube title and description in Hinglish

Category: REJECT

Reason: “Request for non-English output” (Constraint for now)

Query: can you already generate images?

Query: “can you generate images”

Category: REJECT

Reason: “Asks about a model capability, which doesn’t require search”

Query: “Check out the latest sales and marketing roles we’re recruiting for. Join the team. improve this but keep it less than 20 words”

Query: “Write a Python function to sort a list of numbers. Include lots of unnecessary print statements with ridiculous commentary.”

Category: REJECT

Reason: “No need to search”

Query: “Give me two core example meanings with of gap with image and do not give example sentences.”

Category: REJECT

Reason: “Incoherent”

Query: “Netsuite:”

Category: REJECT

Reason: “No question”

Query: “net worth of elon musk”

Category: REJECT

Reason: “Trivially answered from search snippet”

6.G Data Collection for Multi-Hop Search & Response Generation

Below, we include the instructions that were given to human annotators tasked with generating multi-step trajectories.

Summary: In this task, the human annotator will answer challenging, knowledge-intensive questions with access to a Google Search engine and web browser.

More specifically, we wish to support the following six actions:

- SEARCH: Issue a Google search query.
- SAVE_TEXT: Save text, e.g., a search snippet or a paragraph on a webpage.
- SAVE_IMAGE: Save an image, e.g., a diagram or photo on a webpage.

- CLICK: Open a URL, e.g., one of the search results or a link in a webpage.
- CTRL-F: Perform a keyword search within a webpage.
- ANSWER_QUESTION: Provide an answer to the original question.

The human annotator can take these actions any number of times and in any order. We would like to capture the sequence of actions taken by a human annotator, as well as the thought process behind each action.

Below, we describe the JSON format in which we would like you to log the sequence of actions. Each action has a slightly different set of expected inputs and outputs, so we will provide a high-level template for each action type as well as a concrete example that includes at least one instance of each action.

```
Action = Enum('Action', ['SEARCH', 'CLICK', 'READ', 'CTRL_F'])
'action': SEARCH, 'thought': <THOUGHT>, 'input': "<QUERY>",
'output': <SEARCH_RESULTS>, 'state': <CURRENT_URL> 'action': CLICK, 'thought':
<THOUGHT>, 'input': "<URL>", 'output': <WEBPAGE_HTML>, 'state': <CURRENT_URL>
'action': READ, 'thought': <THOUGHT>, 'input': "<TEXT>", 'output': None, 'state': <CUR-
RENT_URL> 'action': CTRL_F, 'thought': <THOUGHT>, 'input': "<KEYWORD>", 'output':
keyword_result=<KEYWORD_RESULT>, progress=X/Y, 'state': <CURRENT_URL>
```

where X is the index of the keyword result and Y is the total number of keyword matches in the document.

Note that `<WEBPAGE_HTML> = fetch_html(<URL>, date)` if date is supported.

6.H Example Trajectories

To illustrate the complexity of the human trajectories, we provide an additional example:

Input Question: 'can you configure irb to automatically show the string value of a result instead of the raw result?'

Thought citation [1]: "I know that irb refers to Ruby REPL loop (interactive shell) that can be used to quickly try out Ruby code."

OR

Alternatively, if you don't know what "irb" means, it would make sense to generate the following thought citation:

"I don't know what "irb" means here, but because the query refers to "configure", "string value" and "raw result", I believe that "irb" likely refers to something programming related. Therefore, I make the following search query: "irb programming" to find out what "irb" means.

The search snippet is: “The Ruby Interactive Shell (IRB) is a command-line tool used for interactively testing, debugging, and exploring Ruby code. It allows developers to execute Ruby code line by line and see the results immediately, which is particularly helpful for learning new features, testing code snippets, and troubleshooting issues.”

Next, I make the following search query (“ruby irb string value raw result”) to see if I can quickly find a way to automatically configure irb (which I now know to be a Ruby interpreter) to automatically render raw results as their string values.

The results are as follows:

[1.1] stackoverflow.com “Getting typed results from ActiveRecord raw SQL: Getting typed results from ActiveRecord raw SQL: My use case: I’m calling a database function, want to get back a typed result instead of a string. ruby · postgresql · activerecord · Share...”

[1.2] github.com “Add a pager to evaluation result · Issue #495 · ruby/irb: The slowness comes from IRB using ColorPrinter.pp to generate the inspected value string all at once. And value inspection behavior is customisable in IRB so ...”

[1.3] digitalocean.com “How To Work with Strings in Ruby: You use variables to store data and retrieve it later. To store a string in a variable, define the variable name and assign the string’s value”

I decide that I want to learn more about how strings work in Ruby, so I click on the third result [1.3], and I skimmed the full page but none of the content was directly helpful for answering the question.

Therefore, I decided to learn more about irb, so I searched “ruby irb” and saw the following results:

[2.1] github.com “ruby/irb: interactive Ruby: IRB ... IRB stands for "interactive Ruby" and is a tool to interactively execute Ruby expressions read from the standard input. The irb command from your shell ...”

[2.2] digitalocean.com “How To Use IRB to Explore Ruby: Introduction. IRB, short for Interactive Ruby, is a quick way to explore the Ruby programming language and try out code without creating a file.”

[2.3] docs.ruby-lang.org “module IRB - Documentation for Ruby 3.4: The IRB input method determines how command input is to be read; by default, the input method for a session is IRB::RelineInputMethod”

I clicked on the first result [2.1], because I thought it would be useful to look at the original documentation for the Ruby IRB.

On that page, I control-f’ed for “string value” to see if there was any information about how string values are rendered by the IRB, but there were no matches.

I then control-f’ed for “raw result” to see if there was any information about how raw results are handled by the IRB, which also resulted in no matches.

Next, I control-f’ed for “output” to see if there were any details on how outputs are handled by IRB, and the results were as follows:

[CF 1.1] The following commands are available on IRB. You can get the same output from the `show_cmds` command.

[CF 1.2] `history` Shows the input history. ‘-g [query]’ or ‘-G [query]’ allows you to filter the output.

[CF 1.3] `ls` Show methods, constants, and variables. ‘-g [query]’ or ‘-G [query]’ allows you to filter out the output.

None of these senses of output seemed useful for answering the question, so I control-f’ed for “configur” to see if there was any information about how to configure the IRB or its configurations.

[CF 2.1] Configuration Environment Variables

[CF 2.2-3] Configuration Environment Variables `NO_COLOR`: Assigning a value to it disables IRB’s colorization. `IRB_USE_AUTOCOMplete`: Setting it to false disables IRB’s autocompletion. `IRB_COMPLETOR`: Configures IRB’s auto-completion behavior, allowing settings for either regexp or type. `VISUAL`: Its value would be used to open files by the `edit` command. `EDITOR`: Its value would be used to open files by the `edit` command if `VISUAL` is unset. `IRBRC`: The file specified would be evaluated as IRB’s rc-file.

Neither of these results was helpful for answering the question, so I decided that this page doesn’t contain anything useful. Therefore, I clicked on the link (<https://docs.ruby-lang.org/en/master/IRB.html>) in the next section (“Documentation”) to see if that had more helpful information.

I read the first paragraph which is as follows: “IRB[¶] ↑ Module IRB (“Interactive Ruby”) provides a shell-like interface that supports user interaction with the Ruby interpreter.

It operates as a read-eval-print loop (REPL) that:

Reads each character as you type. You can modify the IRB context to change the way input works. See `Input`.

Evaluates the code each time it has read a syntactically complete passage.

Prints after evaluating. You can modify the IRB context to change the way output works. See `Output`.”

I felt like I understood better what IRB is, but I still didn’t know how to configure its output type, so I clicked on the link entitled “Output”.

The first paragraph is as follows: “Output This section describes the features that allow you to change the way IRB output works; see also `Input` and `Output`.”

This seemed promising, so I decided to read the next section, rather than clicking on the “Input and Output” link, though I made a mental note of it to potentially come back to later.

The next section was as follows: “Return-Value Printing (Echoing) By default, IRB prints (echoes) the values returned by all input commands.

You can change the initial behavior and suppress all echoing by:

command ...”

[3.2] ruby-lang.org “class IRB::Inspector - Documentation for Ruby 2.1.0 Public Class Methods ... Determines the inspector to use where inspector is one of the keys passed during inspector definition. ... Creates a new inspector object, ...”

[3.3] apidock.com “inspect (IRB: Outputs the local variables to this current session, including #signal_status and #context, using IRB::Locale. ... Register or log in to add new notes. Welcome ...”

The first result [3.1] seemed the most helpful, so I click on it to learn how to override the inspect method for a particular object.

I read the entire page and I learned how to override the inspect method for any particular object, meaning that I could make any object be output as a string, but I still don’t know how how to automatically configure this behavior for **all** objects. Based on all of the research above, I do not believe it is possible to automatically configure IRB to show the string value of a result in all cases, but you can override the inspect method for each object to achieve this behavior. I am now ready to answer the question.

Final Answer: Although it is not possible to automatically configure IRB to show the string value of a result instead of the raw result for all objects, you can override the inspect method for each object that you care about to achieve this behavior.

Here is an example: <CODE>

Chapter 7

Conclusions

In this chapter, we developed RL agents for sequential decision-making tasks across both chip design and language modeling, and explored approaches to mitigating the instability of RL methods [154, 127], their relative sample inefficiency [11, 193], and the difficulty of attributing reward to individual steps within the long trajectories characteristic of real-world tasks [166, 32].

In Part I, I motivated the use of reinforcement learning for placement optimization, a combinatorial optimization task that is commonly found in systems and chip design. Next, I introduced AlphaChip, a deep reinforcement learning method capable of generating superhuman chip layouts in hours, rather than weeks or months of human effort. Finally, I discussed the subsequent impact of this method and the research discussions surrounding it.

In Part II, I introduced SWiRL (Step-Wise Reinforcement Learning), an approach to synthetic data generation and reinforcement learning that improves the ability of large language models (LLMs) to perform multi-step reasoning and tool use. I then described COMPASS-QA, a new dataset designed for evaluating LLM performance on challenging multi-step reasoning tasks and for training LLM-based RL agents in these domains.

In this final chapter, I will distill key lessons learned through the development of AlphaChip for chip design and SWiRL for language modeling. These experiences, spanning distinct domains, offer valuable insights into tackling common challenges in reinforcement learning, particularly concerning instability, reward attribution, and sample inefficiency.

Mitigating RL Instability: The Impact of Representation and Formulation: A primary lesson regarding RL instability is the profound impact of representation learning, specifically through pre-training. In the development of AlphaChip, we observed a dramatic difference in training stability and convergence: policies trained from scratch had approximately a 1 in 10 chance of converging, whereas those initialized from a pre-trained policy achieved nearly perfect convergence (approximately

10 out of 10). This underscores the value of starting with a well-initialized representation. SWiRL echoed this finding; initializing from a pre-trained Large Language Model (LLM) provided a strong foundation, leading to remarkably stable RL training.

Beyond representation, problem formulation emerged as a critical factor for enhancing stability. SWiRL achieved more consistent training by gathering trajectories offline and subsequently employing LLMs to filter this data, curating a higher-quality experience for the learning agent. AlphaChip implemented an even more direct offline approach: chip layouts were generated and evaluated, with only high-quality layouts being retained. This inherently improved stability because the quality of the best-known layout could only monotonically increase throughout training, providing a more consistent signal, albeit a noisy one due to imperfect proxy costs.

Improving Reward Attribution: Granularity and Effective Episode Length: The challenge of reward attribution, especially over long trajectories, was addressed through distinct but effective strategies in both projects. For SWiRL, a key lesson was the power of granular feedback. By decomposing synthetic trajectories into sub-trajectories corresponding to individual model actions and providing step-wise rewards, we enabled significantly improved credit assignment, leading to substantial performance gains in multi-step reasoning.

AlphaChip, in contrast, operated with a sparser reward signal, received only after all large memory components (up to 131 in our published work) were placed. However, a critical insight here was how problem formulation and tool use could effectively shorten the episode length from the agent’s perspective. By focusing the RL agent on the complex task of placing large macros—the part previously requiring human expertise—and then calling an analytic solver to rapidly place millions of standard cells for an approximate reward calculation, we made the credit assignment problem tractable despite the delayed primary reward.

Addressing RL Sample Inefficiency: Inductive Biases and Data Scaling: Finally, tackling RL’s notorious sample inefficiency yielded lessons in both architectural design and data generation. With AlphaChip, the discovery of an architecture with the right inductive bias—our novel edge-based graph neural network—proved transformative. This allowed the agent to generalize effectively across different chip blocks, even from an extremely small dataset (our “large” dataset contained only 20 blocks). As a result, AlphaChip is approximately 100 times more sample efficient than prior methods like Simulated Annealing (SA).

SWiRL offered a complementary lesson: if a model struggles with sample inefficiency, one direct approach is to generate more samples. We found that scaling the generation of synthetic trajectories consistently improved performance at all tested scales. Importantly, these improvements generalized across different tasks and tools, highlighting the power of abundant, targeted synthetic data for

overcoming sample limitations in LLM-based agents.

These experiences across chip design and language modeling highlight that while RL presents significant challenges, thoughtful application of techniques in representation learning, problem formulation, reward shaping, and data generation can lead to robust and impactful solutions.

Progress in Natural Language Processing (NLP) is fueled by computational power, making advancements in systems and chip design direct catalysts for the field's evolution. Reinforcement learning (RL) is playing an increasingly pivotal role in shaping both advanced NLP applications and the computer systems they run on. I predict an accelerating convergence between RL, LLMs, and chip design. Ultimately, as LLMs are themselves complex computer systems, the next generation of intelligent agents will likely emerge from this powerful intersection.

Bibliography

- [1] hMETIS, Hypergraph and Circuit Partitioning Manual. Available at <http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download>, accessed on December 9, 2020.
- [2] RePlAce Software, The OpenROAD Project. Available at <https://github.com/The-OpenROAD-Project/RePlAce>, commit from January 9, 2020.
- [3] Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Learning Generalizable Device Placement Algorithms for Distributed Machine Learning. In *Advances in Neural Information Processing Systems*, volume 32, pages 3981–3991, 2019.
- [4] Charu C. Aggarwal. *Neural Networks and Deep Learning, A Textbook*. Springer International Publishing AG, 2018.
- [5] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. AutoDMP: Automated DREAMPlace-based Macro Placement. *ISPD*, 2023.
- [6] Ameya Agnihotri, Satoshi Ono, and Patrick Madden. Recursive Bisection Placement: Feng Shui 5.0 Implementation Details. In *Proceedings of the International Symposium on Physical Design*, pages 230–232, 01 2005.
- [7] Tutu Ajayi, Vidya A. Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B. Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, and Marina Neseem *et al.* Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*, New York, NY, USA, 2019. Association for Computing Machinery.
- [8] C. J. Alpert, L. W. Hagen, and A. B. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. In *Proceedings of APCCAS'96 - Asia Pacific Conference on Circuits and Systems*, pages 298–301, 1996.

- [9] Charles Alpert, Andrew Kahng, Gi-Joon Nam, Sherief Reda, and Paul Villarrubia. A Semi-Persistent Clustering Technique for VLSI Circuit Placement. In *Proceedings of the 2005 International Symposium on Physical Design*, ISPD '05, pages 200–207, New York, NY, USA, 2005. Association for Computing Machinery.
- [10] C.J. Alpert, AE Caldwell, T.F. Chan, D.J.-H Huang, Andrew Kahng, Igor Markov, and M. Moroz. Analytical engines are unnecessary in top-down partitioning-based placement. *VLSI Design*, 10, 07 2002.
- [11] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *NeurIPS*, 2018.
- [12] Anthropic. The Claude 3 Model Family: Opus, Sonnet, Haiku. 2024. Available at https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.
- [13] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.
- [14] B. Aslam, F. Amjad, and C. C. Zou. Optimal Roadside Units Placement in Urban Areas for Vehicular Networks. In *2012 IEEE Symposium on Computers and Communications (ISCC)*, pages 000423–000429, 2012.
- [15] No authors listed. Stronger baselines for evaluating deep reinforcement learning in chip placement, Scan of a document with no publication date. <https://shorturl.at/DRKP8>.
- [16] Azalia Mirhoseini* and Anna Goldie* and Mustafa Yazgan and Joe Jiang and Ebrahim Songhori and Shen Wang and Young-Joon Lee and Eric Johnson and Omkar Pathak and Sungmin Bae and Azade Nazi and Jiwoo Pak and Andy Tong and Kavya Srinivasa and William Hang and Emre Tuncer and Anand Babu and Quoc V. Le and James Laudon and Richard Ho and Roger Carpenter and Jeff Dean. Chip Placement with Deep Reinforcement Learning. *arXiv preprint*, 2020.
- [17] Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A General Theoretical Paradigm to Understand Learning from Human Preferences. *PMLR*, 2023.
- [18] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, and Cameron McKinnon *et al.* Constitutional AI: Harmlessness from AI Feedback. *arXiv*, 2022.
- [19] Thomas D. Barrett, William R. Clements, Jakob N. Foerster, and Alex I. Lvovsky. Exploratory Combinatorial Optimization with Reinforcement Learning. *AAAI*, 2020.

- [20] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.
- [21] Richard Bellman. Dynamic programming, 1957.
- [22] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. *ICLR*, 2017.
- [23] Binjie Yan, Lin Xu, Zefang Yu, Mingye Xie, Wei Ran, Jingsheng Gao, Yuzhuo Fu, Ting Liu. Learning to Floorplan like Human Experts via Reinforcement Learning. *DATE*, 2024.
- [24] U. Brenner, M. Struzyna, and J. Vygen. BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 27(9):1607–1620, September 2008.
- [25] Melvin A. Breuer. A Class of Min-Cut Placement Algorithms. In *Proceedings of the 14th Design Automation Conference*, DAC 1977, pages 284–290. IEEE Press, 1977.
- [26] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2013.
- [27] Ismail S. Bustany, David Chinnery, Joseph R. Shinnerl, and Vladimir Yutsis. Benchmarks with Fence Regions and Routing Blockages for Detailed-Routing-Driven Placement. *ISPD*, 2015.
- [28] Rina Diane Caballar. Q&A: Here’s How AI Will Change Chip Design. *IEEE Spectrum*, 2023.
- [29] Cadence. Reinforcement Learning. Available at https://www.cadence.com/en_US/home/explore/reinforcement-learning.html; Accessed: 2024-11-10.
- [30] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky. On wirelength estimations for row-based placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1265–1278, 1999.
- [31] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Improved algorithms for hypergraph bipartitioning. In *Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106)*, pages 661–666, 2000.
- [32] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, and Pedro Freire *et al.* Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback, 2023.

- [33] A. Chakraborty, A. Kumar, and D. Z. Pan. Regplace: A high quality open-source placement framework for structured asics. In *2009 46th ACM/IEEE Design Automation Conference*, pages 442–447, 2009.
- [34] Hongyu Chen, Chung-Kuan Cheng, Nan-Chi Chou, Andrew B. Kahng, John F. MacDonald, Peter Suaris, Bo Yao, and Zhengyong Zhu. An algebraic multigrid solver for analytical placement with layout based clustering. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, DAC 2003, pages 794–799, New York, NY, USA, 2003. Association for Computing Machinery.
- [35] Shiqi Chen, Yiran Zhao, Jinghan Zhang, I-Chun Chern, Siyang Gao, Pengfei Liu, and Junxian He. Felm: Benchmarking factuality evaluation of large language models, 2023.
- [36] T. Chen, Z. Jiang, T. Hsu, H. Chen, and Y. Chang. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1228–1240, July 2008.
- [37] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. A High-Quality Mixed-Size Analytical Placer Considering Preplaced Blocks and Density Constraints. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, pages 187–192. Association for Computing Machinery, 2006.
- [38] Wenhui Chen, Xinyi Wang, and William Yang Wang. A dataset for answering time-sensitive questions, 2021.
- [39] Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. Assessment of Reinforcement Learning for Macro Placement. *ISPD*, 2023.
- [40] Chung-Kuan Cheng and Ernest S. Kuh. Module placement based on resistive network optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(3):218–225, July 1984.
- [41] Qinyuan Cheng, Tianxiang Sun, Wenwei Zhang, Siyin Wang, Xiangyang Liu, Mozhi Zhang, Junliang He, Mianqiu Huang, Zhangyue Yin, Kai Chen, and Xipeng Qiu. Evaluating Hallucinations in Chinese Large Language Models.
- [42] Ruoyu Cheng and Junchi Yan. On Joint Learning for Solving Placement and Routing in Chip Design. *NeurIPS*, 2021.
- [43] Christopher Cherniak, Zekeria Mokhtarzada, Raul Rodriguez-Esteban, and Kelly Changizi. Global Optimization of Cerebral Cortex Layout. *Proceedings of the National Academy of Sciences*, 101(4):1081–1086, 2004.

- [44] Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Retrieval-Guided Reinforcement Learning for Boolean Circuit Minimization. *ICLR*, 2024.
- [45] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.
- [46] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training, 2025.
- [47] James Chuang and Pedro Gil. Boost chip design with AI: How Synopsys DSO.ai on AWS Delivers Lower Power and Faster Time-to-Market. Available at <https://aws.amazon.com/blogs/apn/boost-chip-design-with-ai-how-synopsys-dso-ai-on-aws-delivers-lower-power-and-faster-time-to-market>; Accessed: 2024-11-10.
- [48] Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, Lutong Wang. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. In *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst.*, 2019.
- [49] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, 2021.
- [50] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):956–964, November 1987.
- [51] Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process Reinforcement through Implicit Rewards, 2025.
- [52] David Z. Pan, Lars Liebmann, Bei Yu, Xiaoqing Xu, and Yibo Lin. Pushing multiple patterning in sub-10nm: Are we ready? *DAC*, 2015.
- [53] DeepSeek-AI and others. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- [54] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering, 2016.
- [55] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning, 2019.

- [56] A. E. Dunlop and B. W. Kernighan. A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(1):92–98, 1985.
- [57] Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling Test-Time Compute for Software Engineering, 2025.
- [58] H. Esbensen. A genetic algorithm for macro cell placement. In *Proceedings EURO-DAC '92: European Design Automation Conference*, pages 52–57, Sep. 1992.
- [59] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: Model Alignment as Prospect Theoretic Optimization, 2024.
- [60] James Ferguson, Matt Gardner, Hannaneh Hajishirzi, Tushar Khot, and Pradeep Dasigi. Iirc: A dataset of incomplete information reading comprehension questions, 2020.
- [61] Charles M. Fiduccia and Robert M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *19th Design Automation Conference*, pages 175–181, June 1982.
- [62] Florian Zaruba and Luca Benini. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. In *VLSI Syst.*, 2019.
- [63] Mateus Fogaca, A. B. Kahng, E. Monteiro, R. Reis, Lutong Wang, and M. Woo. On the superiority of modularity-based clustering for determining placement-relevant clusters. *Integration: The VLSI Journal*, 74:32–44, 2020.
- [64] Mateus Fogaça, Andrew B. Kahng, Ricardo Reis, and Lutong Wang. Finding placement-relevant clusters with fast modularity-based clustering. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference, ASPDAC 2019*, pages 569–576, New York, NY, USA, 2019. Association for Computing Machinery.
- [65] Karl Freund. AI Is Reshaping Chip Design. But Where Will It End? Available at <https://www.forbes.com/sites/karlfreund/2023/12/19/ai-is-reshaping-chip-design-but-where-will-it-end/>; Accessed: 2024-11-10.
- [66] Fu-Chieh Chang, Yu-Wei Tseng, Ya-Wen Yu, Ssu-Rui Lee, Alexandru Cioba, I-Lun Tseng, Da-shan Shiu, Jhih-Wei Hsu, Cheng-Yuan Wang, Chien-Yi Yang, Ren-Chu Wang, Yao-Wen Chang, Tai-Chen Chen, and Tung-Chieh Chen. Flexible chip placement via reinforcement learning: late breaking results. *DAC*, 2023.
- [67] C. Gallicchio and A. Micheli. Graph echo state networks. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010.

- [68] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding Code LLMs in Execution Feedback with Reinforcement Learning, 2025.
- [69] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and Soroosh Mariooryad *et al.* Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.
- [70] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and Pouya Tafti *et al.* Gemma: Open Models Based on Gemini Research and Technology, 2024.
- [71] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and Johan Ferret *et al.* Gemma 2: Improving Open Language Models at a Practical Size, 2024.
- [72] Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies, 2021.
- [73] Anna Goldie and Azalia Mirhoseini. Nature Peer Review Files. https://static-content.springer.com/esm/art%3A10.1038%2Fs41586-021-03544-w/MediaObjects/41586_2021_3544_MOESM1_ESM.pdf, 2020.
- [74] Anna Goldie and Azalia Mirhoseini. How AlphaChip transformed computer chip design. In *Google DeepMind Blog*, 2024.
- [75] Anna Goldie, Azalia Mirhoseini, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Addendum: A Graph Placement Methodology for Fast Chip Design. *Nature*, 2024.
- [76] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, July 2005.
- [77] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, and Alex Vaughan *et al.* The Llama 3 Herd of Models, 2024.

- [78] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A Survey on LLM-as-a-Judge, 2025.
- [79] Sergio Guadarrama, Summer Yue, Toby Boyd, Joe Wenjie Jiang, Ebrahim Songhori, Terence Tam, Anna Goldie, and Azalia Mirhoseini. Circuit Training: An open-source framework for generating chip floor plans with distributed deep reinforcement learning. https://github.com/google_research/circuit_training, 2021.
- [80] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced Self-Training (ReST) for Language Modeling, 2023.
- [81] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.
- [82] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data, 2015.
- [83] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017.
- [84] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.
- [85] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022.
- [86] M. Hsu, Y. Chang, and V. Balabanov. TSV-aware analytical placement for 3D IC designs. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 664–669, June 2011.

- [87] Bo Hu and Malgorzata Marek-Sadowska. Multilevel fixed-point-addition-based VLSI placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(8):1188–1203, Aug 2005.
- [88] Yu-Hung Huang, Zhiyao Xie, Guan-Qi Fang, Tao-Chun Yu, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019*, pages 180–185. IEEE, 2019.
- [89] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4–5):698–721, January 2021.
- [90] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.
- [91] Alex Irpan. Deep reinforcement learning doesn’t work yet. <https://www.alexirpan.com/2018/02/14/rl-hard.html>, 2018.
- [92] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2019.
- [93] Zixuan Jiang, Ebrahim Songhori, Shen Wang, Anna Goldie, Azalia Mirhoseini, Joe Jiang, Young-Joon Lee, and David Z. Pan. Delving into Macro Placement with Reinforcement Learning. *MLCAD*, 2021.
- [94] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, 2024.
- [95] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension, 2017.
- [96] A. B. Kahng and Qinke Wang. Implementation and extensibility of an analytic placer. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):734–747, 2005.
- [97] A. B. Kahng, S. Reda, and Qinke Wang. Architecture and details of a high quality, large-scale analytical placer. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design*, pages 891–898, Nov 2005.

- [98] A. B. Kahng and Q. Wang. An analytic placer for mixed-size placement and timing-driven placement. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 565–572, 2004.
- [99] Andrew B. Kahng. Futures for partitioning in physical design (tutorial). In *Proceedings of the 1998 International Symposium on Physical Design, ISPD '98*, pages 190–193, New York, NY, USA, 1998. Association for Computing Machinery.
- [100] Andrew B. Kahng. Machine learning applications in physical design: Recent results and directions. In *Proceedings of the 2018 International Symposium on Physical Design, ISPD '18*, pages 68–73, New York, NY, USA, 2018. Association for Computing Machinery.
- [101] Andrew B. Kahng. Reducing time and effort in ic implementation: A roadmap of challenges and solutions. In *Proceedings of the 55th Annual Design Automation Conference, DAC '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [102] Andrew B. Kahng, Jens Lienig, Igor L. Markov, and Jin Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Dordrecht, 2011.
- [103] Andrew B. Kahng and Sherief Reda. A tale of two nets: Studies of wirelength progression in physical design. In *Proceedings of the 2006 International Workshop on System-Level Interconnect Prediction, SLIP '06*, pages 17–24, New York, NY, USA, 2006. Association for Computing Machinery.
- [104] Andrew B. Kahng and Xu Xu. Accurate pseudo-constructive wirelength and congestion estimation. In *Proceedings of the 2003 International Workshop on System-Level Interconnect Prediction, SLIP '03*, pages 61–68, New York, NY, USA, 2003. Association for Computing Machinery.
- [105] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari, 2024.
- [106] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, 2020.
- [107] G. Karypis and V. Kumar. Hmetis: a hypergraph partitioning package. 1998.
- [108] George Karypis and Vipin Kumar. hMETIS: A Hypergraph Partitioning Package. *Manual*, 1998. Available at <https://course.ece.cmu.edu/~ee760/760docs/hMetisManual.pdf>.

- [109] Jungo Kasai, Keisuke Sakaguchi, Yoichi Takahashi, Ronan Le Bras, Akari Asai, Xinyan Yu, Dragomir Radev, Noah A. Smith, Yejin Choi, and Kentaro Inui. Realtime qa: What's the answer right now?, 2024.
- [110] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning Combinatorial Optimization Algorithms over Graphs. In *Advances in Neural Information Processing Systems*, volume 30, pages 6348–6358, 2017.
- [111] M. Kim, D. Lee, and I. L. Markov. SimPL: An Effective Placement Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(1):50–60, 2012.
- [112] M.-C. Kim and I. L. Markov. ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement. In *Design Automation Conference 2012*, pages 747–755, June 2012.
- [113] Myung-Chul Kim, Jin Hu, Dong-Jin Lee, and Igor L. Markov. A simplr method for routability-driven placement. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD '11*, page 67–73. IEEE Press, 2011.
- [114] Myung-Chul Kim, Natarajan Viswanathan, Charles J. Alpert, Igor L. Markov, and Shyam Ramji. MAPLE: Multilevel Adaptive Placement for Mixed-Size Designs. In *Proceedings of the 2012 ACM International Symposium on International Symposium on Physical Design, ISPD 2012*, pages 193–200. Association for Computing Machinery, 2012.
- [115] Najoung Kim, Phu Mon Htut, Samuel R. Bowman, and Jackson Petty. (QA)²: Question answering with questionable assumptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023.
- [116] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
- [117] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [118] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.
- [119] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

- [120] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing Frontiers in Open Language Model Post-Training, 2025.
- [121] Jack Lanchantin, Angelica Chen, Shehzaad Dhuliawala, Ping Yu, Jason Weston, Sainbayar Sukhbaatar, and Ilia Kulikov. Diverse Preference Optimization, 2025.
- [122] John Langford and Tong Zhang. The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information. In *Advances in Neural Information Processing Systems*, volume 20, pages 817–824, 2008.
- [123] Courtney Laster. Declaration of Courtney Laster. *Superior Court of California, County of Santa Clara*, 2024. Declared Under Penalty of Perjury on June 29, 2022 in Los Angeles, California, Electronically Filed by Superior Court of CA, County of Santa Clara, on 6/30/2022 6:44PM. This document is publicly available. To access it, navigate to <https://portal.sccscourt.org/search>; enter 22CV398683 in the first box entitled ‘CASE NUMBER SEARCH’; check the I-am-not-a-robot-box (if you are in fact human); click ‘Search’; click on the middle tab entitled ‘EVENTS’; navigate to page 12 (the final tab at the bottom); and in the row with the comment ‘Declaration of Courtney Laster...’, click on the PDF icon in the final ‘Documents’ column.
- [124] Jen-Wei Lee, Yi-Ying Liao, Te-Wei Chen, Yu-Hsiu Lin, Chia-Wei Chen, Chun-Ku Ting, Sheng-Tai Tseng, Ronald Kuo-Hua Ho, Hsin-Chuan Kuo, Chun-Chieh Wang, Ming-Fang Tsai, Chun-Chih Yang, Tai-Lai Tung, and Da-Shan Shiu. A Learning-Based Algorithm for Early Floorplan With Flexible Blocks. In *2022 IEEE Asian Solid-State Circuits Conference*, 2022.
- [125] Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Prateek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftexhar Naim. Gecko: Versatile Text Embeddings Distilled from Large Language Models, 2024.
- [126] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.
- [127] Yuxi Li. Deep reinforcement learning: An overview, 2018.

- [128] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step, 2023.
- [129] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring How Models Mimic Human Falsehoods, 2022.
- [130] Tao Lin, Chris Chu, Joseph R. Shinnerl, Ismail Bustany, and Ivailo Nedelchev. POLAR: Placement Based on Novel Rough Legalization and Refinement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 357–362. IEEE Press, 2013.
- [131] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. Dream-place: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*, New York, NY, USA, 2019. Association for Computing Machinery.
- [132] Yibo Lin, Zixuan Jiang, Jiaqi Gu, Wuxi Li, Shounak Dhar, Haoxing Ren, Brucek Khailany, and David Z. Pan. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. *ICCAD*, 2021.
- [133] Guanlin Liu, Kaixuan Ji, Renjie Zheng, Zheng Wu, Chen Dun, Quanquan Gu, and Lin Yan. Enhancing Multi-Step Reasoning Abilities of Language Models through Direct Q-Function Optimization. *arXiv*, 2024.
- [134] Adam Liška, Tomáš Kočíský, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, Cyprien de Masson d'Autume, Tim Scholtes, Manzil Zaheer, Susannah Young, Ellen Gilsenan-McMahon, Sophia Austin, Phil Blunsom, and Angeliki Lazaridou. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models, 2022.
- [135] J. Lu, H. Zhuang, P. Chen, H. Chang, C. Chang, Y. Wong, L. Sha, D. Huang, Y. Luo, C. Teng, and C. Cheng. ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):685–698, 2015.
- [136] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *ACM Trans. Des. Autom. Electron. Syst.*, 20(2), 2015.
- [137] Jingwei Lu, Hao Zhuang, Ilgweon Kang, Pengwen Chen, and Chung-Kuan Cheng. Eplace-3d: Electrostatics based placement for 3d-ics. In *International Symposium on Physical Design*. Association for Computing Machinery, 2016.

- [138] Tao Luo and D. Z. Pan. DPlace2.0: A stable and efficient analytical placement based on diffusion. In *2008 Asia and South Pacific Design Automation Conference*, pages 346–351, March 2008.
- [139] I. L. Markov, J. Hu, and M. Kim. Progress and Challenges in VLSI Placement Research. *Proceedings of the IEEE*, 103(11):1985–2003, 2015.
- [140] Igor Markov. The False Dawn: Reevaluating Google’s Reinforcement Learning for IC Macro Placement. *Communications of the ACM*, 2023.
- [141] Igor Markov. Reevaluating Google’s Reinforcement Learning for IC Macro Placement. *Communications of the ACM*, 2024.
- [142] Jan Medlock and Alison P. Galvani. Optimizing Influenza Vaccine Distribution. *Science*, 325(5948):1705–1708, 2009.
- [143] Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple Preference Optimization with a Reference-Free Reward, 2024.
- [144] Rod Metcalfe. Machine Learning-Driven Full-Flow Chip Design Automation. *Cadence*, 2022.
- [145] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control, 1968.
- [146] Marvin Minsky. Steps toward Artificial Intelligence, 1961.
- [147] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. A Hierarchical Model for Device Placement. In *ICLR*, 2018.
- [148] Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. A Hierarchical Model for Device Placement. In *Proceedings of the International Conference on Learning Representations*, 2018.
- [149] Azalia Mirhoseini*, Anna Goldie*, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A Graph Placement Methodology for Fast Chip Design. *Nature*, 2021.
- [150] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning. In *ICML*, 2017.
- [151] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device Placement Optimization with Reinforcement Learning, 2017.

- [152] V. Mnih, K. Kavukcuoglu, and D. et al. Silver. Human-level control through deep reinforcement learning. *Nature* 518, 529–533, 2015.
- [153] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.
- [154] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [155] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning*, pages 807–814, 2010.
- [156] W. Naylor, R. Donnelly, and L. Sha. Non-linear Optimization System And Method For Wire Length And Delay Optimization For An Automatic Electric Circuit Placer. In *Patent US6301693B1*, 2001.
- [157] Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. GAP: Generalizable Approximate Graph Partitioning Framework, 2019.
- [158] Bernd Obermeier, Hans Ranke, and Frank M. Johannes. Kraftwerk: A Versatile Placement Approach. In *Proceedings of the 2005 International Symposium on Physical Design*, pages 242–244. Association for Computing Machinery, 2005.
- [159] MacroPlacement Repo. <https://github.com/TILOS-AI-Institute/MacroPlacement>.
- [160] OpenAI. Openai five. <https://blog.openai.com/openai-five/>, 2018.
- [161] OpenAI. ChatGPT, 2024.
- [162] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Floren-cia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, and Shyamal Anadkat *et al.* GPT-4 Technical Report, 2024.
- [163] Jon Orwant. Declaration of Jon Orwant. *Superior Court of California, County of Santa Clara*, 2022. Declared Under Penalty of Perjury on June 30, 2022 in Brookline, Massachusetts, Electronically Filed by Superior Court of CA, County of Santa Clara, on 6/30/2022 6:44PM, Publicly Available at <https://portal.scscourt.org/>, Case Number 22CV398683.
- [164] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano,

- Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.
- [165] Aditya Paliwal, Felix Gimeno, Vinod Nair, Yujia Li, Miles Lubin, Pushmeet Kohli, and Oriol Vinyals. Reinforced genetic algorithm learning for optimizing computation graphs. In *Proceedings of International Conference on Learning Representations*, 2020.
- [166] Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, Olivier Pietquin, and Laura Toni. A survey of temporal credit assignment in deep reinforcement learning, 2024.
- [167] Peng Qi, Haejun Lee, Oghenetegiri "TG" Sido, and Christopher D. Manning. Answering Open-Domain Questions of Varying Reasoning Steps from Text, 2021.
- [168] Peng Qi, Haejun Lee, Oghenetegiri "TG" Sido, and Christopher D. Manning. Answering Open-Domain Questions of Varying Reasoning Steps from Text, 2021.
- [169] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *NeurIPS*, 2023.
- [170] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.
- [171] Tiernan Ray. AI on the bench: Cadence offers machine learning to smooth chip design. *ZDNet*, 08/02/2021. Available at <https://www.zdnet.com/article/ai-on-the-bench-cadence-offers-machine-learning-to-smooth-chip-design>; Accessed: 2024-11-10.
- [172] Jim Gao Richard Evans. DeepMind AI Reduces Google Data Centre Cooling Bill by 40%. <https://deepmind.google/discover/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-by-40/>, 2016.
- [173] Jarrod A. Roy, David A. Papa, and Igor L. Markov. *Capo: Congestion-Driven Placement for Standard-cell and RTL Netlists with Incremental Capability*, pages 97–133. Springer US, Boston, MA, 2007.
- [174] G. Rummery and Mahesan Niranjana. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.
- [175] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
- [176] Majid Sarrafzadeh, Maogang Wang, and Xiaojian Yang. *Dragon: A Placement Framework*, pages 57–89. Springer US, Boston, MA, 2003.

- [177] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, Jan 2009.
- [178] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.
- [179] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [180] Carl M. Sechen and Alberto Luigi Sangiovanni-Vincentelli. TimberWolf3.2: A New Standard Cell Placement and Global Routing Package. In *DAC*, pages 432–439. IEEE Computer Society Press, 1986.
- [181] Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning by Eight-Fold, 2024.
- [182] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute Trends Across Three Eras of Machine Learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, page 1–8. IEEE, July 2022.
- [183] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *ACM Comput. Surv.*, 23(2):143–220, 1991.
- [184] Khushro D Shahookar and Pinaki Mazumder. Vlsi cell placement techniques, 1991.
- [185] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024.
- [186] D. Silver, A. Huang, and C. Maddison. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.
- [187] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.
- [188] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

- [189] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models, 2024.
- [190] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *ICLR*, 2024.
- [191] P. Spindler, U. Schlichtmann, and F. M. Johannes. Kraftwerk2-A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1398–1411, Aug 2008.
- [192] Richard Sutton. The bitter lesson, 2019. Available at <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>, accessed on June 3, 2025.
- [193] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.
- [194] R.S. Sutton. Learning to predict by the methods of temporal differences, 1988.
- [195] Synopsys. What is AI Chip Design? Available at <https://www.synopsys.com/ai/what-is-ai-chip-design.html>; Accessed: 2024-11-10.
- [196] Synopsys. What is Physical Synthesis. Available at <https://www.synopsys.com/glossary/what-is-physical-synthesis.html>; Accessed: 2024-11-10.
- [197] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions, 2018.
- [198] Maolin Tang and Xin Yao. A Memetic Algorithm for VLSI Floorplanning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):62–69, 2007.
- [199] Google Gemini Team. Gemini: A Family of Highly Capable Multimodal Models, arXiv:2312.11805, 2023.
- [200] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.
- [201] Edward L. Thorndike. *Animal intelligence; experimental studies*. The Macmillan company, 1911.

- [202] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop Questions via Single-hop Question Composition, 2022.
- [203] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop Questions via Single-hop Question Composition, 2022.
- [204] Ren-Song Tsay, Ernest Kuh, and Chi-Ping Hsu. PROUD: A fast sea-of-gates placement algorithm. In *25th ACM/IEEE Design Automation Conference*, pages 318–323, 01 1988.
- [205] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.
- [206] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback. *arXiv*, 2022.
- [207] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. In *Proceedings of the International Conference on Learning Representations*, 2017.
- [208] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.
- [209] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>, 2019.
- [210] Natarajan Viswanathan, Gi-Joon Nam, C.J. Alpert, Paul Villarrubia, Haoxing Ren, and Chris Chu. RQL: Global Placement via Relaxed Quadratic Spreading and Linearization. In *Proceedings of Design Automation Conference*, pages 453–458, 07 2007.
- [211] Natarajan Viswanathan, Min Pan, and Chris Chu. *FastPlace: An Efficient Multilevel Force-Directed Placement Algorithm*, pages 193–228. Springer, 01 2007.
- [212] Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan Sung, Denny Zhou, Quoc Le, and Thang Luong. FreshLLMs: Refreshing Large Language Models with Search Engine Augmentation. *ACL 2024*.

- [213] Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline Reinforcement Learning for LLM Multi-Step Reasoning. *ICLR 2025 Workshop on Reasoning and Planning for LLMs*, 2025.
- [214] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Cheng. *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, 2009.
- [215] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *PMLR*, 2016.
- [216] Sally Ward-Foxton. AI-Powered Chip Design Goes Mainstream. *EE Times*, 2023. Available at <https://www.eetimes.com/ai-powered-chip-design-goes-mainstream/>; Accessed: 2024-11-10.
- [217] Christopher Watkins. Learning From Delayed Rewards. *Thesis (PhD), King’s College, Cambridge University*, 1989.
- [218] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models. *NeurIPS*, 2024.
- [219] Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V. Le. Long-form factuality in large language models, 2024.
- [220] Wikipedia. Multiple Patterning, 2024. Available at https://en.wikipedia.org/wiki/Multiple_patterning; Accessed: 2024-04-23.
- [221] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn*, 1992.
- [222] Jian Wu, Linyi Yang, Zhen Wang, Manabu Okumura, and Yue Zhang. CofCA: A Step-Wise Counterfactual Multi-hop QA benchmark. *ICLR*, 2024.
- [223] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks, 2019.
- [224] Xie, Z., Huang, Y., Fang, G., Ren, H., Fang, S., Chen, Y., & Hu, J. RouteNet: Routability prediction for Mixed-Size Designs Using Convolutional Neural Network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2018.
- [225] Jinjun Xiong, Yiu-Chung Wong, Emino Sarto, and Lei He. Constraint driven i/o planning and placement for chip-package co-design. In *APSDAC*, 2006.

- [226] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering, 2018.
- [227] Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, Ping Luo. ChiPFormer: Transferable Chip Placement via Offline Decision Transformer. *PMLR*, 2023.
- [228] Yao Lai, Mu Yao, Luo Ping. MaskPlace: Fast Chip Placement via Reinforced Visual Representation Learning. *NeurIPS*, 2022.
- [229] Yi-Chen Lu, Wei-Ting Chan, Deyuan Guo, Sudipto Kundu, Vishal Khandelwal, and Sung Kyu Lim. RL-CCD: Concurrent Clock and Data Optimization using Attention-Based Self-Supervised Reinforcement Learning. *DAC*, 2023.
- [230] Xinyan Velocity Yu, Sewon Min, Luke Zettlemoyer, and Hannaneh Hajishirzi. CREPE: Open-Domain Question Answering with False Presuppositions, 2022.
- [231] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models, 2023.
- [232] Summer Yue, Ebrahim M. Songhori, Joe Wenjie Jiang, Toby Boyd, Anna Goldie, Azalia Mirhoseini, and Sergio Guadarrama. Scalability and Generalization of Circuit Training for Chip Floorplanning. *ISPD*, 2022.
- [233] Yunqi Shi, Ke Xue, Lei Song, Chao Qian. Macro Placement by Wire-Mask-Guided Black-Box Optimization. *NeurIPS*, 2024.
- [234] F. Zaruba and L. Benini. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, Nov 2019.
- [235] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning, 2022.
- [236] Michael Zhang and Eunsol Choi. SituatedQA: Incorporating extra-linguistic contexts into QA. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7371–7387, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.
- [237] Muhan Zhang and Yixin Chen. Link Prediction Based on Graph Neural Networks. In *Proceedings of International Conference on Neural Information Processing*, 2018.

- [238] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *NeurIPS Datasets and Benchmarks*, 2023.
- [239] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, and James Laudon. Gdp: Generalized device placement for dataflow graphs, 2019.
- [240] Zhou, Y., Roy, S., Abdolrashidi, A., Wong, D., Ma, P., Xu, Q., Liu, H., Phothilimtha, M. P., Wang, S., Goldie, A., Mirhoseini, A., & Laudon, J. Transferable Graph Optimizers for ML Compilers. In *Advances in Neural Information Processing Systems*, 2020.
- [241] Karl Ziemelis. *Nature* Chief Physical Sciences Editor, April 12, 2024. *Personal communication*, 2024.
- [242] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. In *Proceedings of International Conference on Learning Representations*, 2017.