Exploring Search for FPGA Placement using RL

Shang Wang[1, 2], **Owen Randall**[1, 2], **Matthew E. Taylor**[1, 2, 3]
{shang8, davidowe, mtaylor3}@ualberta.ca

[1]**Department of Computing Science, University of Alberta, Canada**
[2]**Alberta Machine Intelligence Institute (Amii)**
[3]**CIFAR AI Chair**

Abstract

Reinforcement learning (RL) has become increasingly popular, but applying it successfully to real-world problems remains challenging. In contrast, search has been a powerful yet underused tool in real-world settings, despite achieving notable successes in several domains. We believe the potential of search-based approaches in real-world applications has not yet been fully explored, as many real-world applications are combinatorial in nature.

Our work explores the use of reinforcement learning guided Monte Carlo tree search (MCTS) algorithms, using AlphaZero-style agents in the challenging real-world task of field-programmable gate array (FPGA) component placement. Our preliminary results show that MCTS can significantly improve a reinforcement learning agent's performance in this task. This, in itself, is not surprising. However, additional results show that by integrating Gumbel-enhanced MCTS, the policy converges faster and achieves better performance, demonstrating the utility of search-based approaches in this real-world application. Our results highlight a broader insight: search is not an outdated baseline — when used in combination with RL methods, it could be an under-used approach for solving real-world decision problems.

1 Introduction

In recent years, deep reinforcement learning (RL) has become a popular paradigm for decision-making tasks, from game playing to robotics (Arulkumaran et al., 2019; Kober J, 2013). Although often overshadowed by recent advances in deep RL, search has achieved notable success in domains such as matrix multiplication optimization (Fawzi et al., 2022) and video encoding (Mandhane et al., 2022). The AlphaZero algorithm (Silver et al., 2018) and its related variations (Silver et al., 2016; 2017; Schrittwieser et al., 2020) have led to revolutionary breakthroughs by combining search and reinforcement learning, allowing for newfound success in complex combinatorial games such as Go. In the domain of chip design, deep RL has recently gained traction as a way to assist and improve traditional placement methods (Lin et al., 2019). Notably, Google introduced a model-free PPO-based framework for macro placement in ASIC chips (Mirhoseini et al., 2021), which inspired substantial follow-up work (Lu et al., 2023; Shi et al., 2023; Lai et al., 2023; 2022; Cheng & Yan, 2021; Chang et al., 2022; Yan et al., 2024).

In this paper, we set out to explore the potential of alternative search-driven approaches, which we believe are better suited for the combinatorial problem of chip design. We use Monte Carlo tree search (Browne et al., 2012) guided by an RL-trained policy to find desirable FPGA chip designs. Our preliminary results show that our search-based approach outperforms a model-free PPO-based baseline in terms of placement quality, measured by estimated wirelength. Furthermore, we experiment with using the Gumbel AlphaZero technique to speed up convergence, and find that it results

in additional performance improvements. We believe that this work supports the idea that search can be a powerful tool for improving reinforcement learning techniques, particularly for complex real-world tasks with combinatorial structure.

2 Background and Related Work

2.1 RL Background

Many real-world decision-making problems, including those in RL, can be modeled as Markov decision processes (MDPs). An MDP is defined by a tuple $\langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, $P(s'|s, a)$ is the transition probability to the next state s' given current state s and action a, R is the reward function, and γ is the discount factor for future rewards. The objective is to learn a policy that maximizes the expected cumulative reward over time.

2.2 PPO

Proximal policy optimization (PPO) (Schulman et al., 2017) is a widely used on-policy reinforcement learning algorithm known for its stability and simplicity. It updates the policy using a clipped objective function, which limits how much the new policy can deviate from the old one, thus stabilizing learning. Its ease of implementation and robust performance have made it a common technique in RL research.

2.3 AlphaZero

The AlphaZero algorithm (Silver et al., 2018) combines a neural network optimization with MCTS (Browne et al., 2012) to guide decision making through search and learning. The policy network provides action priors and value estimates, which MCTS uses to iteratively construct a tree of states. Each iteration of MCTS consists of four phases:

1. **Selection:** Starting from the root node (current state), the tree is traversed using the PUCT formula, which balances exploitation (selecting high-value nodes) and exploration (selecting low-visit nodes) (Kocsis & Szepesvári, 2006; Rosin, 2010).
2. **Expansion:** When a leaf node is reached, a new node is added to the tree to represent the next possible state.
3. **Evaluation:** The policy and value network evaluate the new node to provide a prior distribution over actions and an estimated return.
4. **Backpropagation:** The evaluation information is propagated back up the tree to update visit counts and action values.

During training, episodes are generated from self-play with multiple MCTS iterations and selecting the most visited action at each step. After completing an episode, the outcome is used to train the policy via cross-entropy loss and the value network via mean squared error between predicted and actual returns from rollouts.

2.4 Gumbel AlphaZero

To enhance AlphaZero's search efficiency, the Gumbel AlphaZero work (Danihelka et al., 2022) introduces various improvements upon the action selection mechanisms of AlphaZero by integrating Gumbel-top-k sampling, sequential halving, and value interpolation. These improvements make Gumbel AlphaZero particularly effective under a limited simulation budget. At the root node, actions are sampled without replacement using the Gumbel-top-k trick (Kool et al., 2019). The top-k root node actions (by Gumbel-perturbed log-probability) are selected, then sequential halving (Karnin et al., 2013) allocates simulations among them, repeatedly discarding the worst half of the remaining actions until one action remains, which is chosen as the root action. For nodes other

than the root: 1) Unvisited node actions are evaluated using value interpolation, enabling a complete policy update; and 2) Visited node actions are selected deterministically based on the visit count distribution, ensuring theoretical policy improvement. This approach allows Gumbel AlphaZero to perform well even with limited simulations, unlike traditional AlphaZero, which requires extensive exploration.

2.5 FPGA Placement

The chip placement problem maps a list detailing a circuit's components and connections (known as the netlist), to a physical layout of the components on the chipboard. The number of FPGA components can range from hundreds to tens of thousands for complex high-density designs, illustrating the scale and variability of the placement challenge. The placement of these components must respect the following two constraints: *Type constraints* specify which components can be placed on different parts of the chipboard. *Capacity constraints* specify the number of components allowed.

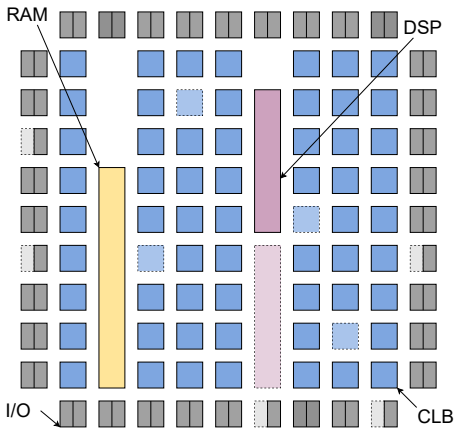

Figure 1: This FPGA board is 11×11 units in size and incorporates components: digital signal processors, configurable logic blocks, I/O blocks, and random access memory blocks.

Figure 1 shows an example diagram of an FPGA chip. The possible component positions are represented by colored rectangles, where each color corresponds to a specific component type.

The most common goal for FPGA placement is to minimize wirelength and critical path delay. In this work, we focus on estimated wirelength as our primary evaluation metric, as it provides a proxy for placement quality and is computationally inexpensive. We estimate wirelength from placed component locations in our experiments, allowing faster iteration while still reflecting meaningful placement differences. Full FPGA design tools such as VTR (Verilog-to-Routing) (Elgammal et al., 2025) can generate the routed wirelength, a more accurate estimate, but require much more simulation time.

2.6 RL-based Placement

AlphaChip (Mirhoseini et al., 2021) was a pioneering work that modeled ASIC macro placement as an RL problem, using a model-free PPO-based agent to learn placement policies. The authors proposed a two-stage pipeline, where a model-free RL agent places the large macro components in the first stage, and a traditional placement tool handles the remaining components in the second stage. The approach inspired a wave of research exploring state representations, offline learning methods, and improved exploration strategies (Cheng & Yan, 2021; Cheng et al., 2022; Lai et al., 2022; 2023). However, the majority of these works remain purely model free and do not leverage search. A notable exception is EfficientPlace (Geng et al., 2024), which combines PPO with MCTS, highlighting the promise of integrating search with RL for chip placement.

Despite this progress, the broader class of RL-based placement methods still rarely incorporate search explicitly. For instance, RLPlace (Elgammal et al., 2021) enhances classical simulated annealing by using a bandit-style RL agent to guide local move strategies. It was later integrated into the VTR framework (Elgammal et al., 2025), where it remains a key component of the state-of-the-art simulated annealing-based placement engine. The Divide-and-Conquer RL approach (Wang et al., 2024) breaks the placement process into sequential subproblems, replying purely on model-free learning.

3 Experiments

This section presents preliminary results using RL to tackle FPGA placement problems, highlighting how advantages in search can significantly improve performance.

3.1 Problem Setup

The FPGA placement problem is formulated as an MDP. Details are in Appendix 5.1, but a sketch is as follows:

- **States** consist of the placement status of the current board.
- **Actions** are defined as placing the current component in a location that does not violate any hard constraints (e.g., capacity or type limits).
- **Rewards** are zero until all components are placed, at which point the final reward is proportional to the estimated wirelength.

3.2 Placement Setup

We evaluate PPO, AlphaZero, and Gumbel AlphaZero agents on simple FPGA placement tasks, with the goal of comparing model-free RL and search-based RL. We use the `tseng.net` netlist and `EArch.xml` architecture files from MCNC20 benchmark suite (Yang, 1991), a standard dataset in FPGA design. The full placement instance consists of 56 CLBs and 174 I/O components. From Table 1, we can see that the number of possible placements grows exponentially with the number of components to be placed, showing the difficulty of this task. In our experiments the RL agent is trained to place either 5 or 15 components, while the remaining components remain at the positions specified by the full VTR solution. Details are in Appendix 5.3.

# Components to place	HPWL by random placement	Placement permutations
5	~ 3400	$\sim 10^5$
15	~ 4300	$\sim 10^{17}$
56	~ 5700	$\sim 10^{83}$
230	~ 7300	$\sim 10^{259}$

Table 1: The number of components to be placed, the approximate average HPWL (i.e., wirelength) resulting from random placement, and the approximate number of possible placement permutations shown for different FPGA placement tasks from the MCNC20 dataset.

3.3 Agents Setup

All agents use a common ResNet-based neural network to encode the current chipboard state as a spatial feature map. The encoded state representation is used as the input to the agent's learned networks, which output a probability distribution over actions (policy head) and an estimate of the expected cumulative return of the state (value head). For the value head, PPO agent outputs a scalar value, while AlphaZero and Gumbel AlphaZero follows the MuZero categorical value head (Schrittwieser et al., 2020). To ensure FPGA placement constraints are met, invalid actions are disalowed (i.e., masked) during both inference and training.

In our initial experiments with the AlphaZero agent, we observed that the MCTS algorithm often gets stuck in local optima, leading to highly skewed visitation distributions where most simulations are concentrated on a small number of state-action pairs. This limited exploration and suppressed the agent's ability to discover better trajectories. However, setting the exploration constant of the PUCT formula (Rosin, 2010) too high degrades performance from excessive exploration from nodes with well-known values. Ideally, we would want much higher exploration on nodes with few visits, and lower exploration on states with sufficiently many visits. To address this, we introduced a forced

exploration mechanism: if the most visited child under the root exceeds a dynamic threshold $N = \frac{k \cdot n_{sim}}{n_{valid\_actions}}$, and at least 20% of the root's children are under this threshold, we randomly select one under-visited node from the top 20% (by PUCT score). The scalar k controls the sensitivity of this threshold, and we set $k = 2.0$ in our experiments. This explicit exploration mechanism encourages visits to promising but underexplored actions, helping the agent escape suboptimal regions and improve trajectory diversity. We have found that this method of exploration in combination with PUCT leads to improved performance for MCTS on this task.

4 Results

We report the training performance of PPO, AlphaZero, and Gumbel AlphaZero on the 5-components and 15-components FPGA placement tasks. Details are in Appendix 5.2.

Figure 2 shows the HPWL curves for each agent during the training in the 5-components and 15-components tasks. Datapoints are the best HPWL found over the episodes generated at that training step, not the best HPWL found over the entire training process. There are two main observations:

1. **Starting point differences:** Each agent completes their first learning step at different times due to varying computational overheads. PPO, being model-free, starts training and generating results fastest, while AlphaZero and Gumbel AlphaZero require additional time for search simulations before producing results.
2. **Search-based agents performance:** Given the same training time, both AlphaZero and Gumbel AlphaZero outperform PPO, achieving better placement quality than PPO. Gumbel AlphaZero not only finds lower HPWL faster, but also shows significantly less variance across different configurations. On the 5-component task, Gumbel AlphaZero was able to quickly converge to the VTR solution across all variations.

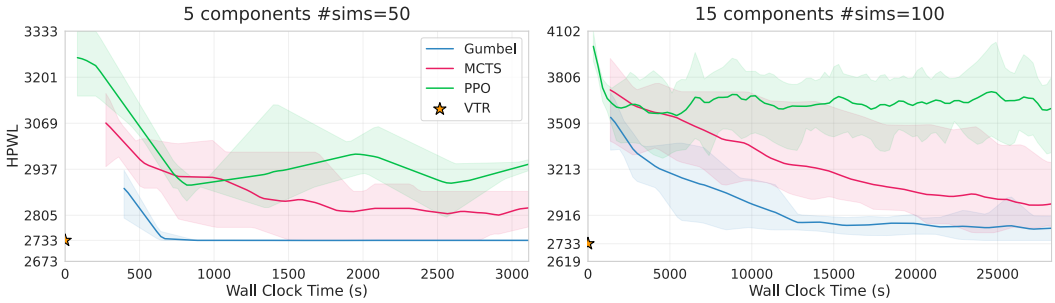

Figure 2: Training performance comparison of PPO, AlphaZero, and Gumbel AlphaZero agents on the 5-component (left) and 15-component (right) FPGA placement tasks. The search agents are given a simulation budget of 50 for the 5 component task and 100 for the 15 component task. The y-axis shows HPWL, and the x-axis shows wall-clock training time in seconds. The VTR-generated placement HPWL ($\sim$ 2 seconds) is marked with an orange star as a reference. Each curve represents the mean across five random configurations, with the range between the highest and lowest HPWL found across each configuration.

Table 2 shows the average HPWL (with standard deviation) across 5 different placement configurations for each agent evaluated on the 5-component and 15-component tasks. We load the checkpoint that achieved the best training HPWL and test it on the same component configuration but with varied simulation budgets. While this setup does not test generalization to unseen configurations, it enables a fair comparison of how each agent uses additional simulation resources at inference. From the table, we observe that with a simulation budget of 1, both AlphaZero and Gumbel AlphaZero yield performance comparable to PPO. As the simulation budget increases, both the average HPWL improves and the standard deviation decreases, demonstrating that search-based inference yields better and more stable placements.

Metric	# Components	Simulation budget	PPO	AlphaZero	Gumbel AlphaZero
HPWL	5	1	2899.60 ± 62.60	2954.87 ± 70.66	2805.70 ± 43.76
		50		2866.80 ± 66.78	2769.95 ± 22.85
		100		2858.14 ± 59.76	2760.63 ± 27.76
	15	1	3582.05 ± 178.70	3534.56 ± 269.96	3377.02 ± 259.43
		50		3300.76 ± 186.56	3121.31 ± 252.54
		100		3204.38 ± 173.93	3091.93 ± 206.22
Time (s)	5	1	0.50 ± 0.02	3.11 ± 0.02	3.18 ± 0.02
		50		6.39 ± 0.02	6.42 ± 0.02
		100		9.48 ± 0.03	9.96 ± 0.05
	15	1	0.83 ± 0.02	3.23 ± 0.02	4.61 ± 0.02
		50		13.60 ± 0.03	13.84 ± 0.11
		100		24.21 ± 0.09	24.71 ± 0.20

Table 2: Average HPWL and runtime (± standard deviation) for PPO, AlphaZero, and Gumbel AlphaZero evaluated on five placement configurations for both 5- and 15-component tasks. Results correspond to the best-performing checkpoint from training. Each method is tested under varying simulation budgets to assess how effectively it balances placement quality and computational cost during inference.

While PPO maintains consistently low inference time across simulation budgets, both AlphaZero and Gumbel AlphaZero incur additional computational cost due to MCTS. As expected, their runtimes increase with the number of simulations, reflecting the overhead of search-based inference. Notably, Gumbel AlphaZero still achieves better HPWL than AlphaZero at lower budgets, suggesting more efficient use of simulations.

5 Conclusion and Future Work

This work explored how integrating search with reinforcement learning, specifically using AlphaZero and Gumbel AlphaZero can enhance performance on real-world inspired combinatorial tasks. Using FPGA placement as a case study, we show that combining search with RL leads to faster convergence and improved solution quality, even under limited simulation budgets. Future work includes extending this direction to more realistic placement metrics and larger problem scales, as well as leveraging recent works in search-augmented policy improvement Pirnay et al. (2023); Randall et al. (2024) to further test the generality of our insights.

Acknowledgments

This work has taken place in the Intelligent Robot Learning (IRL) Lab at the University of Alberta, which is supported in part by research grants from the Alberta Machine Intelligence Institute (Amii); a Canada CIFAR AI Chair, Amii; Digital Research Alliance of Canada; Huawei; Mitacs; and NSERC.

Appendix

5.1 State Construction

We define each state as an image composed of the following six channels:

1. The capacity channel indicates the remaining capacity of each grid cell.
2. The action mask channel indicates the locations where the current components can be placed, with a value of 1 for valid positions and 0 for invalid ones.
3. The input channel indicates the number of times the placed components serve as a source/input in all nets.
4. The output channel indicates the number of times the placed components serve as a sink/output in all nets.
5. The output channel indicates the number of connections the placed components has with other components in the netlist.
6. The wire-mask channel represents how the estimated wirelength increases if a component is placed in a position (Lai et al., 2022).

Each channel is a matrix of the same size as the chipboard, and together they comprise all relevant information pertaining to the current state of the chipboard and the components to be placed. The image composed of these channels is used as the input for our learned policy. This format should only be used if there is a single appendix (unlike in this document).

5.2 Training Setup

All training has been conducted on a Compute Canada cloud server equipped with 1x Tesla V100-SXM2-16GB GPU and Intel Xeon Gold 6148 CPU at 2.40GHz. For the 5-components task, we train each agent for 3600 seconds (1 hour), using a simulation budget of 50 per step for AlphaZero and Gumbel AlphaZero agent. For the 15-components task, we train each agent for 28800 seconds (8 hours), using a simulation budget of 100 per step for AlphaZero and Gumbel AlphaZero agent.

To evaluate the consistency of the agent's performance, we construct 5 different component placement task configurations for each setting (5-components and 15-components), each defined by a randomly sampled subset of CLBs. For each configuration, the agent is tasked with placing the selected components, while all other components remain fixed to their VTR-optimal positions. The agent places the components in a predefined, fixed order, eliminating the additional complexity of learning placement sequences.

5.3 Placement Setup

To create a more tractable and interpretable setting, we decided to simplify the experiments. We assume that a placement generated by VTR serves as an expert solution, and we fix the positions of all components except for a small subset. The RL agent is then trained to place either 5 or 15 components, while the remaining components remain at their VTR-optimal positions. We evaluate performance using estimated half-perimeter wirelength (HPWL) and report the gap between the VTR-optima and the RL placement results.

To provide a consistent and bounded learning signal, we define the reward as a linearly scaled version of the HPWL, mapped to the range $(-1, 0)$. For example, in the 5-components task, we map HPWL values ranging from 3400 (random placement) to 2733 (VTR-derived placement) linearly to the interval such that 3400 maps to -1 and 2733 maps to 0. We apply the same reward scaling to the 15-components task, mapping (4300, 2733) to (-1, 0).

References

Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. In *Proceedings of the genetic and evolutionary computation conference companion*, pp. 314–315, 2019.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. DOI: 10.1109/TCIAIG.2012.2186810.

Fu-Chieh Chang, Yu-Wei Tseng, Ya-Wen Yu, Ssu-Rui Lee, Alexandru Cioba, I-Lun Tseng, Dashan Shiu, Jhih-Wei Hsu, Cheng-Yuan Wang, Chien-Yi Yang, et al. Flexible chip placement via reinforcement learning: late breaking results. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1392–1393, 2022.

Ruoyu Cheng and Junchi Yan. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 34:16508–16519, 2021.

Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems*, 35:26350–26362, 2022.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

Mohamed A Elgammal, Kevin E Murray, and Vaughn Betz. Rlplace: Using reinforcement learning and smart perturbations to optimize fpga placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2532–2545, 2021.

Mohamed A Elgammal, Amin Mohaghegh, Soheil Gholami Shahrouz, Fatemehsadat Mahmoudi, Fahrican Koşar, Kimia Talaei, Joshua Fife, Daniel Khadivi, Kevin Murray, Andrew Boutros, et al. Vtr 9: Open-source cad for fabric and beyond fpga architecture exploration. *ACM Transactions on Reconfigurable Technology and Systems*, 2025.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning. *Nature*, pp. 47–53, 2022. DOI: 10.1038/s41586-022-05172-4.

Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Mingxuan Yuan, Jianye Hao, Yongdong Zhang, and Feng Wu. Reinforcement learning within tree search for fast macro placement. In *Forty-first International Conference on Machine Learning*, 2024.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pp. 1238–1246. PMLR, 2013.

Peters J. Kober J, Bagnell JA. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research.*, pp. 1238–1274, 2013. DOI: doi:10.1177/0278364913495721.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML*, pp. 282–293, 2006.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pp. 3499–3508. PMLR, 2019.

Yao Lai, Yao Mu, and Ping Luo. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems*, 35:24019–24030, 2022.

Yao Lai, Jinxin Liu, Zhentao Tang, Bin Wang, Jianye Hao, and Ping Luo. Chipformer: Transferable chip placement via offline decision transformer. In *International Conference on Machine Learning*, pp. 18346–18364. PMLR, 2023.

Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

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. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2023.

Amol Mandhane, Anton Zhernov, Maribeth Rauh, Chenjie Gu, Miaosen Wang, Flora Xue, Wendy Shang, Derek Pang, Rene Claus, Ching-Han Chiang, Cheng Chen, Jingning Han, Angie Chen, Daniel J. Mankowitz, Jackson Broshear, Julian Schrittwieser, Thomas Hubert, Oriol Vinyals, and Timothy Mann. Muzero with self-competition for rate control in vp9 video compression, 2022.

A. Mirhoseini, A. Goldie, M. Yazgan, and et al. A graph placement methodology for fast chip design. *Nature*, pp. 207—212, 2021. DOI: 10.1038/s41586-021-03544-w.

Jonathan Pirnay, Quirin Göttl, Jakob Burger, and Dominik Gerhard Grimm. Policy-based self-competition for planning problems. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=SmufNDN90G.

Owen Randall, Martin Müller, Ting-Han Wei, and Ryan Hayward. Expected work search: combining win rate and proof size estimation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1. DOI: 10.24963/ijcai.2024/774.

Christopher Rosin. Multi-armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence*, 61:203–230, 2010. DOI: 10.1007/s10472-011-9258-6.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. Macro placement by wire-mask-guided black-box optimization. *Advances in Neural Information Processing Systems*, 36:6825–6843, 2023.

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529:484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Driessche, Thore Graepel, and Demis Hassabis. Mastering the Game of Go without Human Knowledge. *Nature*, 550:354–359, 2017. DOI: 10.1038/nature24270.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. DOI: 10.1126/science.aar6404.

Shang Wang, Deepak Ranganatha Sastry Mamillapalli, Tianpei Yang, and Matthew E. Taylor. Fpga divide-and-conquer placement using deep reinforcement learning. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, pp. 690–696, 2024. DOI: 10.1109/ISEDA62518.2024.10617677.

Binjie Yan, Lin Xu, Zefang Yu, Mingye Xie, Wei Ran, Jingsheng Gao, Yuzhuo Fu, and Ting Liu. Learning to floorplan like human experts via reinforcement learning. In *2024 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–2. IEEE, 2024.

Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. Citeseer, 1991.