2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)
Phoenix, Arizona, USA, August 6-9, 2023

AI-Enhanced Codesign of Neuromorphic Circuits

Douglas C. Crowder*, J. Darby Smith, Suma G. Cardwell
Cognitive and Emerging Computing
Sandia National Laboratories
Albuquerque, NM
*dccrowd@sandia.gov

2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-0210-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/MWSCAS57524.2023.10405942

Abstract—Leading neuromorphic computer (NMC) platforms achieve energy efficiency and extreme scalability by implementing simplified models of biological neurons. Future NMCs that rely on complex neuron models for complex calculations will require labor-intensive design processes that are currently performed by highly-trained professionals. In this work, we develop AI-enhanced tools for automating NMC design, demonstrating that such tools can design next-generation circuits.

Index Terms—reinforcement learning, evolutionary algorithms, electronic design automation, neuromorphic computers, AI-enhanced codesign

I. Introduction

Neuromorphic computers (NMCs) are energy-efficient computing systems, inspired by the biological brain, [1] that can efficiently implement spiking neural networks. These include analog, digital, mixed-signal, and beyond-CMOS implementations. Due to their energy efficiency and small footprint, NMCs are useful in applications where size, weight and power (SWaP) are constrained. NMCs have been successfully utilized in embedded systems, scientific computing, and artificial intelligence [2]–[4].

Currently, most NMCs implement simplified spiking dynamics, such as the leaky integrate-and-fire (LIF) model [5]. Simple NMC LIF neurons capture the quintessential spiking activity of biological neurons. However, they fail to capture important and complex computational functions present in biological neurons such as dendritic processing, and learning synapses. For instance, neuronal dendrites perform non-linear filtering, coincidence detection, and other sub-threshold computations that are sensitive to spatio-temporal patterns of inputs [6].

NMCs that support dendrite-like computations could perform more complex functions with fewer computing elements [7]–[10]. Thus, dendritic NMC elements could be highly useful for SWaP constrained applications, such as remote signal detection. However, the design of NMCs is highly non-trivial, requiring the use of expert knowledge. To accelerate the design of advanced NMCs, we propose to use design automation methods that rely on artificial intelligence (AI).

AI has previously been used for many electronic design automation (EDA) tasks, including: device placement, connection routing, component sizing, digital logic design, software to hardware mapping, SPICE simulation, lithography, device modeling, runtime management, high-level synthesis, and transistor sizing [11]–[19]. AI has also been used to optimize photonic machine learning accelerators [20]. Additionally, evolutionary algorithms (EAs) have been used to optimize NMC circuits [21]. However, because EAs do not perform inference, they may have difficulty in scaling to large problems. In contrast, reinforcement learning (RL) agents can perform inference, and techniques such as curriculum learning and transfer learning can allow RL to efficiently scale to more difficult problems.

Here, we demonstrate RL agents that are able to learn to design dendrite-like analog NMC circuits [8], [22] that can perform signal discrimination. During training, the RL agents place NMC components into a circuit and tune component parameters to successfully complete signal discrimination tasks. Once trained, the RL agents can rapidly design signal discrimination circuits. We compare the results of RL circuit design to EA circuit optimization and demonstrate that both RL and EA methods can be used to rapidly improve computational devices. To the best of our knowledge, this is the first instance of using RL to design circuits for NMCs from scratch, rather than optimizing an existing circuit.

II. Circuit Construction Task

RL and EA methods were evaluated on a task that involved designing low-SWaP circuits from dendrite-inspired components in order to discriminate between signal and noise. The signal (Target) and noise (Non-Target) could be thought of as currents or voltages produced by a sensor. Target and Non-Target samples were drawn from distributions that were not always distinct. This task required the choice of signal detection components or delay components. Detect components attempted to discriminate between Target and Non-Target signals; Delay components

Component Name	Properties & Parameters
Delay	The output of this component is the output of the previous component.
Simple Detect	$G \in [-10, 10]$, $B = 0$, $L = 0$
Leaky Detect	$G \in [-10, 10]$, $B = 0$, $L = 0.1$
Tunable Leaky Detect	$G \in [-10, 10]$, $B = 0$, $L \in [-10, 10]$
Leaky Biased Detect	$G \in [-10, 10]$, $B \in [-10, 10]$, $L = 0.1$

TABLE I
Dendritic components. G = gain. B = bias. L = Leak.

only propagated their input forward. The dendrite-inspired series circuits were based on the passive dendrite cable model using CMOS transistors as described in [8], [22]; the dendrite components (Figure 1B) abstracted the dendrite sub-circuits [8]. This CMOS-based dendritic circuit models the classical resistor-capacitor (RC) delay line circuit for a passive linear dendrite cable as described by [23]. The CMOS transistor-based model of the dendrite cable models the resistors using transistors that operate in a linear sub-threshold region. The RC behavior is captured using Delay and Leak in our abstracted model. We also model a tunable Bias parameter that offsets the signal value. Dendrites have been shown to have similar multiplicative/gain effects using NMDA (N-methyl-D-aspartate) channels [24] and shunting inhibition [9], and we specify this as the Gain term in our abstracted model.

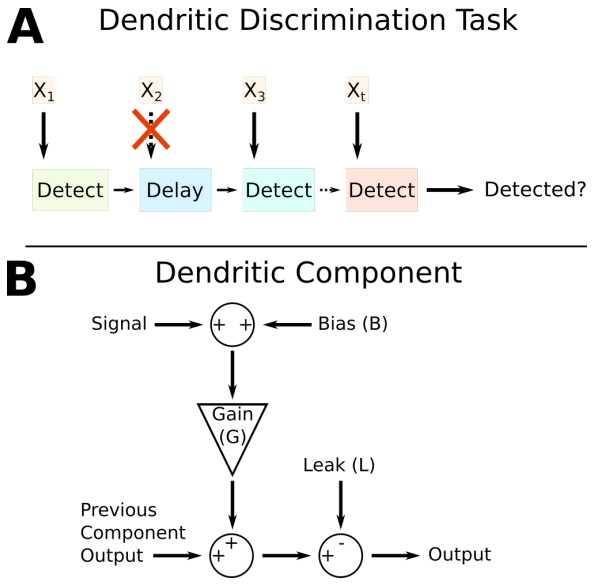

Fig. 1. (A) The agent uses an observation X at each time step to choose a Detect or a Delay component as well as the component parameters. Delay components do not process the signal. Successful circuits detect the Target, but not Non-Target, signals. (B) Diagram of the abstracted dendritic component model.

We created four different models of Detect components based on the dendrite model (see Table I). Each model had a tunable or fixed bias B, gain G, and leakage L. These four models were chosen based on signal detection accuracy results while using the RL circuit design methods (see Section III-B). The agents had to construct a circuit using only Delay components and one type of tunable Detect component.

EA and RL circuit design methods were both provided with signals that contained 10-100 samples from both the Target and the Non-Target distributions (see Figure 2). For six of the samples, the Target distribution was different than the Non-Target distribution. For all remaining samples, the Target distribution was equal to the Non-Target distribution. For those samples where the Target distribution differed, the mean μ was drawn uniformly from $(-0.9, -0.5) \cup (0.5, 0.9)$. Once μ was drawn, the Target signal was drawn uniformly from $\mu \pm 0.1$ The Non-Target distribution was always uniform on $(-1, 1)$.

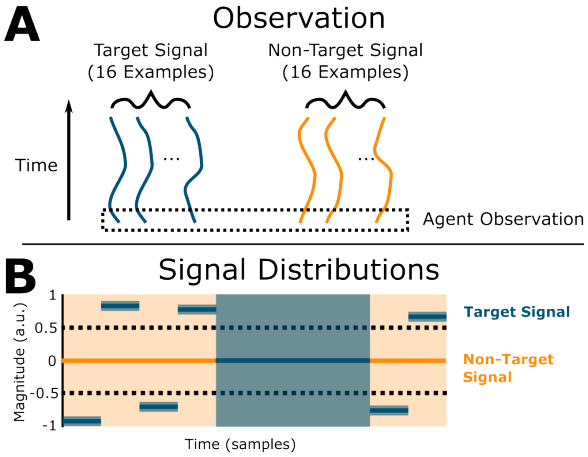

Fig. 2. (A) The RL and EA agents receive 10-100 samples of Target and Non-Target data (in this illustration, 10). (B) The signals are drawn from Target and Non-Target distributions. For six of the samples, the Target distribution is distinct from the Non-Target distribution. Bold lines and shaded regions represent the mean and range of the distributions, respectively. Dotted lines represent lower limit (0.5) for Target signal means. a.u. = arbitrary units.

We chose these Target and Non-Target distributions for two reasons. First, overlapping distributions prevent agents from relying on simple thresholding, instead requiring them to place both Detect and Delay components. Second, as simple thresholding cannot solve this problem, it would be difficult for a human to design such circuits on the first try. Indeed, due to NMC component interaction, a human would likely need to make an initial guess and then tune component parameters. As described in the next section, the RL agent must complete this task by using inference alone, with no additional optimization.

For all experiments, software was implemented in Python 3.6. RL was implemented using stable-baselines version 1.3.0 [25]. Timing experiments were performed on a workstation with an Intel Xeon E5-1650 CPU running at 3.60 GHz and an NVIDIA Quadro P5000 GPU. For RL, 5 agents were trained and then tested on 100 design tasks. For EA, 5 EA optimization processes were performed.

III. Reinforcement Learning

We begin with a basic overview of reinforcement learning as it applies to this task (Section III-A) and then present the results of the RL circuit design (Section III-B).

A. Reinforcement Learning Overview

A cartoon of RL is provided in Figure 3A. In RL [26], an agent observes an environment and uses the observation to choose actions. The actions are applied to the environment, causing the environment to transition to a new state and emit a new observation, as well as a scalar reward. The rewards, observations, and actions are used by a parameter update function to update the parameters of the agent.

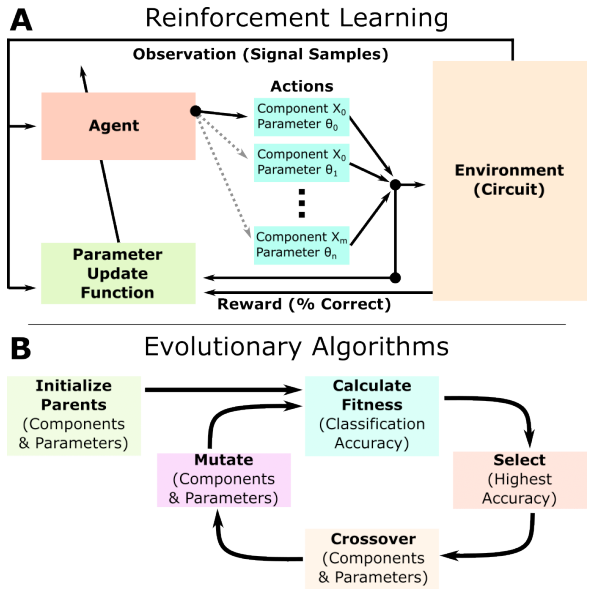

Fig. 3. (A) Overview of RL. (B) Overview of EAs.

1) Proximal policy optimization: We used a deep RL algorithm known as Proximal Policy Optimization (PPO) [27]. In PPO, the agent and the parameter update function both rely on deep neural networks.

2) Observation: At each time step, the RL agent observed 16 samples from the time-varying Target distribution and 16 samples from the time-invariant Non-Target distribution (Figure 2).

3) Action: At each time step, the agent chose a component from the available library. The library consisted of a Delay component and a single type of tunable Detect component (see Table I).

4) Reward: At the terminal time step, the RL agent receiveed a reward that was proportional to the number of correctly classified Target and Non-Target signals:

$$\frac{\#\text{ correct} - \#\text{ incorrect}}{\text{total }\#}.$$

At all non-terminal time steps, the RL agent received a reward of 0. This strategy is referred to as "sparse" because it results in rewards of 0 for most time steps. "Dense" reward strategies provide non-zero rewards at most time steps.

5) Curriculum and Transfer Learning: Dense reward strategies tell RL agents why low-performing designs are "incorrect." However, dense strategies require the investigator to supply information about why a low-performing design is "incorrect," and such information may not be available. Sparse reward strategies can be expected to generalize across circuit design tasks. However, because sparse rewards contain relatively little information, they may cause RL agents to learn slowly.

In order to accelerate RL, we used curriculum learning, progressively increasing the difficulty of the task as the RL agents learned. During learning, signal lengths were progressively increased from 10 samples to 20 samples. At the conclusion of learning, RL agents were tested on circuits that were as long as 100 samples, without retraining (a form of transfer learning).

B. RL Agent Results

1) Accuracy: For the initial task with signal lengths of 10, baseline performance was established using the Simple Detect component. Using the Simple Detect component, the RL agent was able to discriminate Target and Non-Target signals with 91% accuracy. Using the Leaky Detect component and training from scratch, signal discrimination accuracy remained at approximately 91%, suggesting that a small leakage current did not degrade performance. To determine if a tunable leakage current could improve the computational ability of dendrite-like NMC components, the RL agent was trained to use the Tunable Leaky Detect component. Unfortunately, signal discrimination accuracy remained at 91%.

After analyzing the initial results, we determined that signal discrimination was most successful when Target and Non-Target signals had, on average, opposite signs. Thus, we created the Leaky Biased Detect component, which had a tunable bias. After implementing a tunable bias, signal discrimination accuracy improved to 97%.

2) Curriculum and Transfer Learning Results: As described above, we employed curriculum and transfer learning as we increased the number of samples in the signal. As shown in Figure 4A, signal discrimination accuracy was nearly identical for signals with lengths of 10 and 20. However, signal discrimination accuracy decreased slowly from 97% to 90% as signal length increased from 20 to 100 samples (without retraining). We compare the curriculum learning results with EA in the next section.

IV. Evolutionary Algorithms Comparison

EAs (Figure 3B) are optimization algorithms inspired by evolution. Briefly, EAs begin with a set of randomly selected parents, which are collections of parameters to be optimized. The parents are evaluated using a so-called fitness function, which produces a scalar fitness score. The parents with the highest fitness scores are combined in a process called crossover to produce offspring. The offspring parameters are then mutated, and the offspring become the parents of a new generation. This cycle is repeated until the fitness score is maximized.

We designed an EA where the parameter space contained the identities of circuit components, limited to the Delay

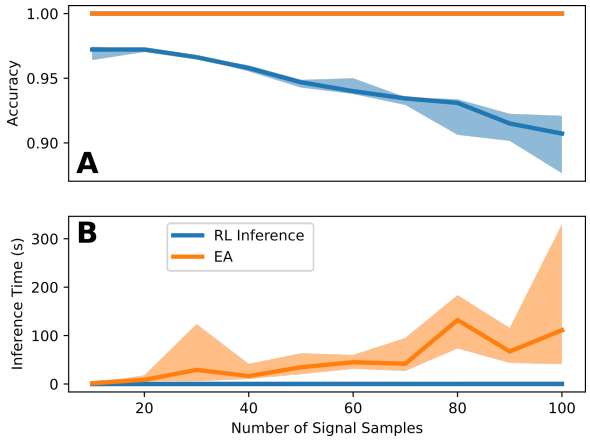

Fig. 4. RL agents were trained to build signal discrimination circuits for signal lengths of 10 samples. Then, signal length was increased to 20 samples using curriculum learning. RL agents were then tested on signal lengths of 20 to 100 samples without retraining. (A) RL signal discrimination accuracy decreased as signal length increased. EA signal discrimination accuracy was unaffected by signal length. (B) Inference time increased linearly with signal length. For RL, inference took only approximately 80 ms for circuits that contained up to 100 elements; EA required more than a minute to optimize similar circuits. Lines and shaded regions represent median ± interquartile range.

and the Leaky Biased Detect components, along with their tunable parameters. The initial population contained 50 individuals. The top 50 individuals in each generation were selected and recombined to produce 50 offspring. Mutations occurred at a rate of $1/N$, where N was the number of samples in the signal. Mutations either caused the type of component to switch from Delay to Detect or caused tunable parameters to be resampled uniformly from $(\gamma - 0.5, \gamma + 0.5)$, where γ represented the existing parameter value. Unlike [21], we assumed a constant circuit topology; only the identities and parameters of components were optimized.

To compare EAs to RL, we considered the time required for finding solutions, as well as the accuracy of the final solution (see Figure 4). The median RL training time was 353 seconds, with an interquartile range of [352, 386] seconds. After training, RL agents could choose 1,301 components per second (median), with a range of [672, 1,325] components per second (Figure 4B). Unlike RL, which uses inference to create designs for a family of problems, EAs are problem specific. Thus, EAs do not have distinct training and inference phases. EAs were able to design circuits at a median rate of 1.39 components per second, with an interquartile range of [0.24, 11.1] components per second. This rate is 2-3 orders of magnitude slower than RL inference, not including the RL training time. However, EAs were able to optimize circuit performance, regardless of signal length, with an accurracy of 100%.

V. Discussion

A. Timing

Advances in NMCs are limited by the availability of human experts, who take years to train. In contrast, AI-enhanced tools, like the one developed here, take relatively little time to train, even when training is performed with inexpensive computing resources. We demonstrated that RL agents can perform simple NMC design tasks in as little as 6 minutes. This rapid training time can be expected to aid in rapidly prototyping novel NMC components. As an example, the RL agent was able to quickly learn that 1) a fixed leakage current does not affect signal discrimination accuracy, 2) a tunable leakage current does not improve signal discrimination accuracy, and 3) a tunable bias does improve signal discrimination accuracy. These 3 facts were determined in less than 24 minutes using widely-available computing resources. Furthermore, after training, RL agents could design NMC circuits for new signal discrimination tasks at a rate of approximately 1,300 components per second, or approximately 0.77 ms per component. Given that humans have reaction times that are on the order of hundreds of milliseconds [28], the current RL agent can design many NMC circuits before a human expert is even aware that there is a design problem. Given that there are no other AI-enhanced tools for designing dendrite-like NMCs, the methods presented here represent the current state-of-the-art.

B. Accuracy

We chose to test the RL agents on an NMC circuit design task that involved discriminating between Target and Non-Target signals that were drawn from overlapping distributions. This task is not only similar to real-world signal discrimination tasks in SWaP-constrained applications, but also very difficult for humans to solve as these problems cannot be solved with simple thresholding. To solve such a problem, a human would likely create an initial design and then optimize the design by tuning parameters – a time-intensive process that would become nearly impossible as the number of NMC components and the number of parameters per component are increased.

We trained an RL agent that was able to successfully complete the task with up to 97% accuracy by using inference alone, without an additional optimization step. Curriculum learning was used to rapidly train the agent to design circuits with up to 20 components without degraded performance. When tested on longer circuits, accuracy degraded slowly from 97% to 90%. It is possible that training the RL agents on the longer circuits could rescue performance back to 97%. Or, alternatively, the NMC design process could be improved by including an explicit optimization step after the initial inference (design) step. Nonetheless, given that the Target and Non-Target signal distributions overlapped, signal discrimination accuracies of 90-97% are quite high, suggesting that these novel RL methods will generalize to many real-world problems.

C. Method Generalizability

We made several important decisions that we expect will allow these methods to generalize to real-world problems.

As discussed above, we chose a task that mimics real-world signal discrimination tasks that are difficult for humans to solve. Additionally, we chose to use sparse rewards that did not directly inform RL agents about how designs could be improved. Sparse reward signals contain relatively little information, which can be expected to impede learning. However, sparse reward signals can always be generated, assuming that circuit performance specifications are known. Thus, we expect that these RL methods will generalize to many interesting real-world tasks. In order to improve the learning rate when using sparse rewards, we chose to use curriculum learning, which allowed us to design fairly complicated NMC circuits.

VI. Conclusions

We developed EA and RL algorithms to automatically construct next-generation, dendrite-like neuromorphic signal discrimination circuits from scratch. This work utilized abstracted models of electrical components for fast circuit design. The RL-designed NMC circuits presented here had simple serial structures. In the future, we will enable RL agents to design NMC circuits with parallel structures and then leverage circuit simulators for further verification and validation.

References

[1] C. Mead, Analog VLSI and Neural Systems. Reading, MA: Addison-Wesley, 1989.

[2] M. Kang, Y. Lee, and M. Park, "Energy efficiency of machine learning in embedded systems using neuromorphic hardware," Electronics, vol. 9, no. 7, 2020.

[3] J. Aimone, P. Date, G. Fonseca-Guerra, K. Hamilton, K. Henke, B. Kay, G. Kenyon, S. Kulkarni, S. Mniszewski, M. Parsa et al., "A review of non-cognitive applications for neuromorphic computing," Neuromorphic Computing and Engineering, 2022.

[4] J. D. Smith, A. J. Hill, L. E. Reeder, B. C. Franke, R. B. Lehoucq, O. Parekh, W. Severa, and J. B. Aimone, "Neuromorphic scaling advantages for energy-efficient random walk computations," Nature Electronics, vol. 5, no. 2, p. 102–112, 2022.

[5] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain et al., "Loihi: A neuromorphic manycore processor with on-chip learning," Ieee Micro, vol. 38, no. 1, pp. 82–99, 2018.

[6] M. London and M. Hausser, "Dendritic computation," Annual Review of Neuroscience, vol. 28, pp. 503–532, Jul. 2005.

[7] J. Acharya, A. Basu, R. Legenstein, T. Limbacher, P. Poirazi, and X. Wu, "Dendritic computing: branching deeper into machine learning," Neuroscience, 2021.

[8] S. George, J. Hasler, S. Koziol, S. Nease, and S. Ramakrishnan, "Low power dendritic computation for wordspotting," Journal of Low Power Electronics and Applications, vol. 3, no. 2, pp. 73–98, 2013.

[9] F. S. Chance and S. G. Cardwell, "Shunting inhibition as a neural-inspired mechanism for multiplication in neuromorphic architectures," in Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference, 2023, pp. 41–46.

[10] S. G. Cardwell and F. S. Chance, "Dendritic computation for neuromorphic applications," in International Conference on Neuromorphic Systems, 2023.

[11] G. Y. Huang, J. B. Hu, Y. F. He, J. L. Liu, M. Y. Ma, Z. Y. Shen, J. J. Wu, Y. F. Xu, H. R. Zhang, K. Zhong, X. F. Ning, Y. Z. Ma, H. Y. Yang, B. Yu, H. Z. Yang, and Y. Wang, "Machine learning for electronic design automation: A survey," Acm Transactions on Design Automation of Electronic Systems, vol. 26, no. 5, 2021, huang, Guyue Hu, Jingbo He, Yifan Liu, Jialong Ma, Mingyuan Shen, Zhaoyang Wu, Juejian Xu, Yuanfan Zhang, Hengrui Zhong, Kai Ning, Xuefei Ma, Yuzhe Yang, Haoyu Yu, Bei Yang, Huazhong Wang, Yu 1557-7309.

[12] H. X. Ren, S. Godil, B. Khailany, R. Kirby, H. G. Liao, S. Nath, J. Raiman, R. Roy, and Ieee, "Optimizing vlsi implementation with reinforcement learning - iccad special session paper," in 40th IEEE/ACM International Conference on Computer Aided Design (ICCAD), ser. ICCAD-IEEE ACM International Conference on Computer-Aided Design, 2021, Conference Proceedings, ren, Haoxing Godil, Saad Khailany, Brucek Kirby, Robert Liao, Haiguang Nath, Siddhartha Raiman, Jonathan Roy, Rajarshi Roy, Rajarshi/0000-0003-4548-2114 1933-7760.

[13] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "Mlcad: A survey of research in machine learning for cad keynote paper," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2021.

[14] A. F. Budak, P. Bhansali, B. Liu, N. Sun, D. Z. Pan, and C. V. Kashyap, "Dnn-opt: An rl inspired optimization for analog circuit sizing using deep neural networks," in 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021, Conference Proceedings, pp. 1219–1224.

[15] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network. in 2020 acm/ieee 2nd workshop on machine learning for cad (mlcad)," 2020.

[16] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement," in International Conference on Learning Representations, 2018.

[17] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022, Conference Proceedings, pp. 27–42.

[18] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi et al., "A graph placement methodology for fast chip design," Nature, vol. 594, no. 7862, pp. 207–212, 2021.

[19] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in 2020 57th ACM/IEEE Design Automation Conference (DAC). IEEE, 2020, pp. 1–6.

[20] Y. Tang, P. T. Zamani, R. Chen, J. Ma, M. Qi, C. Yu, and W. Gao, "Device-system end-to-end design of photonic neuromorphic processor using reinforcement learning," Laser & Photonics Reviews, vol. 17, no. 2, p. 2200381, 2023.

[21] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, "Evolutionary optimization for neuromorphic systems," in Proceedings of the Neuro-inspired Computational Elements Workshop, 2020, pp. 1–9.

[22] S. Nease, S. George, P. Hasler, S. Koziol, and S. Brink, "Modeling and implementation of voltage-mode cmos dendrites on a reconfigurable analog platform," IEEE transactions on biomedical circuits and systems, vol. 6, no. 1, pp. 76–84, 2011.

[23] I. Segev, J. Rinzel, and G. M. Shepherd, The theoretical foundation of dendritic function: the collected papers of Wilfrid Rall with commentaries. The MIT Press, 2003.

[24] Y. Wang and S.-C. Liu, "A two-dimensional configurable active silicon dendritic neuron array," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 58, no. 9, pp. 2159–2171, 2011.

[25] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," Journal of Machine Learning Research, vol. 22, no. 268, pp. 1–8, 2021.

[26] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[28] F. Donders, "On the speed of mental processes," Acta Psychologica, vol. 30, pp. 412–431, 1969.