Self-Organising Digital Circuits

Marcello Barylli[1]*, Gabriel Béna[2]*, Alexander Mordvintsev[3], Eleni Nisioti[1] and Sebastian Risi[1]

[1]IT University of Copenhagen
[2]Imperial College London
[3]Google, Paradigms of Intelligence Team

bary@itu.dk, g.bena21@imperial.ac.uk

Abstract

Fault tolerance in computing has traditionally relied on hardware redundancy and error-correcting codes. Biological intelligence, in contrast, exhibits mechanisms that allow for adaptive resilience. Natural automata like the brain maintain function by re-organising around damaged components. Inspired by this principle, we introduce an architecture for self-organising, fault-tolerant digital circuits. Our hierarchical architecture uses a Graph Transformer (GT) that meta-learns to configure the lookup tables (LUTs) of the circuits. Unlike Neural Cellular Automata (NCAs), which learn to directly generate a target, our network learns an indirect, functional policy. This allows it to *reconfigure* permanently damaged or topologically randomised circuits, rather than relying on regeneration. We demonstrate that our meta-learned network can rapidly reconfigure circuits with previously unseen damage patterns. Our work brings the principles of self-organising systems to the practical domain of digital hardware, where distributed models can offer benefits like compactness and robustness against noise.

Submission type: **Late Breaking Abstract**

Data/Code + **Circuit Visualization** available here

Introduction

Engineered circuits typically achieve fault tolerance through static redundancy, whereas biological systems exhibit graceful degradation through dynamic re-organisation around damage (Nudo, 2013). Inspired by von Neumann's principle of systems that "operate across errors" (von Neumann, 1966), we develop a framework for self-organising digital circuits.

We adapt principles from Neural Cellular Automata (NCAs) and graph variants (Mordvintsev et al., 2020; Grattarola et al., 2021), where cell states typically form the output pattern directly. In our approach, these states are the entries of Boolean Lookup Tables (LUTs) that collectively perform a given computation. The update rules are trained only on the final circuit output (eg binary multiplication of input bits), not on specific LUT target values. This induces an indirect, functional relationship between the self-organising process and the target.

Our work provides a concrete, hardware-adjacent model for ALIFE principles, pushing Graph NCA-like self-organising systems closer to solving practical engineering problems. Its adaptive capacity relies on forward passes only, potentially allowing for on-chip deployment in reprogrammable hardware, such as Field Programmable Gate Arrays (FPGAs) (Carter et al., 1986). This stands in contrast with backpropagation, which requires circuit components to be differentiable (Sunada et al., 2025).

Methods

We use a differentiable circuit representation where each gate is a LUT. Following Petersen et al. (2022), a continuous relaxation of the LUTs enables gradient-based optimisation. Unlike their approach, we use recursive multilinear interpolation to define gate identity, rather than a learned categorical distribution.

We represent each circuit as a graph where nodes correspond to gates and edges to wires. Each node holds the gate's LUT parameters, a hidden state, and positional encodings. This graph structure, defined by the circuit's wiring, constrains the bidirectional message passing in our Graph Transformer (GT) (Yuan et al., 2025). The GT acts as the meta-learner, trained on a persistent pool of circuit instances representing the distribution of optimisation examples (Fig. 1).

Inner Loop (Circuit "Growth" and Reconfiguration): For each circuit in a batch sampled from the pool, the GT performs a forward pass of N message passing steps to compute updates for each node's LUT parameters, analogous to the NCA growth and regeneration steps.

Outer Loop (Meta-Parameter Update): The updated circuits are executed, their outputs are compared against a target Boolean function, and a meta-loss is calculated. This gradient is used to GT's weights, improving its general policy for reconfiguring circuits.

*Denotes equal contribution.

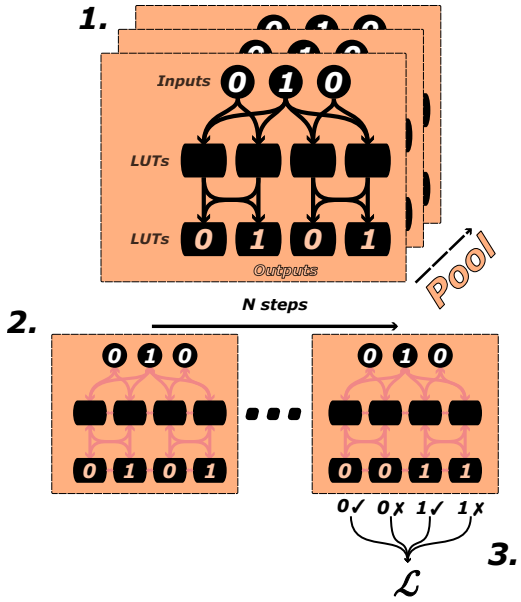

Figure 1: Summary of training framework. ***1.*** Sampling of circuits (with possibly multiple damage or wiring patterns) from the pool. ***2.*** Inner loop: optimization using the GT for message passing between LUTs, updating states. ***3.*** Outer Loop: Functional loss $\mathcal{L}$ is computed by running final optimized circuit. We then backpropagate through the entire inner loop to update the GT's parameters.

Experiments

Wire Shuffles. To illustrate the relationship between structure and function, we implement wire shuffle experiments. The goal is to maintain input-output mapping (functional invariance) across variations of the circuit's wiring topology. Preliminary results show that a single GT optimiser is able to optimise for arbitrary wiring on fixed topologies (number of gates and layers), in the case of simple Boolean task such as *bit-reverse*. Extension to more complicated tasks such as binary multiplication, as well as achieving proper scale-free optimisation (single policy for any topologies), are still under way.

Stuck-at Fault Gate Failures. Another test of functional invariance is component failure. We simulate failure via "stuck-at 0" faults, where a damaged gate's output is permanently clamped to zero and the gate is removed from the message-passing graph on the GT level. This captures a real-world scenario (Mei, 1974) and forces the system to develop a robust strategy for reconfiguring remaining intact gates.

Figure 2 shows such a reconfiguration trajectory. First, a circuit is configured via backpropagation to solve a binary multiplication task. 20 gates are damaged randomly, and the GT performs the reconfiguration.

Error bars indicate performance across 256 damage patterns. On both seen and unseen tasks, the meta-learner is

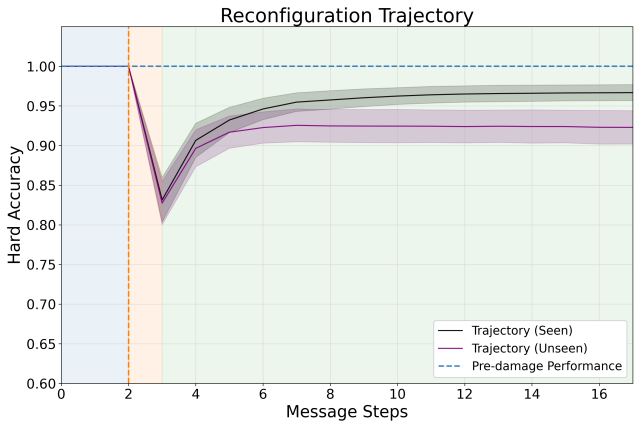

Figure 2: **Reconfiguration Trajectory following damage.** Blue shaded region indicates pre-damage timesteps. The orange dashed line indicates damage time point. Green indicates recovery trajectory timesteps.

capable of recovering a significant but not perfect performance, showcasing graceful degradation (Zhou et al., 2007).

To compare the GT's message passing scheme with recovery via backpropagation, we measure the Hamming distance of the reconfigured circuits for both methods to see which one is further from the unperturbed baseline. The Hamming distance measures in how many positions (ie circuit-wide LUT entries) the reconfigured circuit differs from the baseline in hard mode (Hamming, 1950). Figure 3 shows that the GT achieves comparable accuracy with a lower Hamming distance across 256 damage patterns. Significance testing remains to be done.

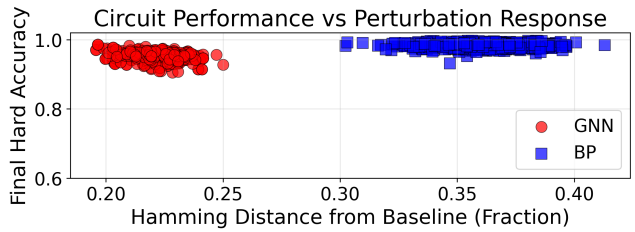

Figure 3: **Measuring Perturbation Response via Hamming Distance**. For remaining undamaged gates, fewer LUT entries are flipped as a reconfiguration following damage for GT than for Backpropagation.

Future Work

We will explore out-of-distribution generalisation by testing larger damage sizes and different circuit topologies. The response to local perturbations will be measured in more detail using visualisations of the circuits and their attention maps. Finally, reversible bit flips, as representing Single Event Upsets, will be repaired.

Acknowledgments

Funded by the European Union (ERC, GROW-AI, 101045094). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council.

References

Carter, W., Duong, I., Freman, R., Hsieh, H., Ja, J., Mahoney, J., Ngo, N., and Sac, S. L. (1986). A user programmable reconfigurable logic array. In *Proceedings of the IEEE Custom Integrated Circuits Conference*.

Grattarola, D., Livi, L., and Alippi, C. (2021). Learning graph cellular automata. arXiv:2110.14237 [cs].

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160.

Mei, K. (1974). Bridging and Stuck-At Faults. *IEEE Transactions on Computers*, C-23(7):720–727.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*, 5(2):e23.

Nudo, R. J. (2013). Recovery after brain injury: mechanisms and principles. *Frontiers in Human Neuroscience*, 7:887.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. (2022). Deep Differentiable Logic Gate Networks. arXiv:2210.08277 [cs].

Sunada, S., Niiyama, T., Kanno, K., Nogami, R., Röhm, A., Awano, T., and Uchida, A. (2025). Blending Optimal Control and Biologically Plausible Learning for Noise-Robust Physical Neural Networks. *Physical Review Letters*, 134(1):017301. Publisher: American Physical Society.

von Neumann, J. (1966). Theory of Self-Reproducing Automata. *University of Illinois Press*.

Yuan, C., Zhao, K., Kuruoglu, E. E., Wang, L., Xu, T., Huang, W., Zhao, D., Cheng, H., and Rong, Y. (2025). A Survey of Graph Transformers: Architectures, Theories and Applications. arXiv:2502.16533 [cs].

Zhou, L., Prabhakaran, V., Ramasubramanian, V., Levin, R., and Thekkath, C. A. (2007). Graceful degradation via versions: specifications and implementations. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, PODC '07, pages 264–273, New York, NY, USA. Association for Computing Machinery.