Binary Image Classification on 2D Partitioned Linear Hybrid Cellular Automata on FPGA

Naoki Sawahashi
Electrical and Computer Engineering
University of Florida
Gainesville, USA
n@oki.ac

Jose C. Principe
Electrical and Computer Engineering
University of Florida
Gainesville, USA
principe@cnel.ufl.edu

***Abstract*—The cellular automaton (CA) is an ideal processor for ultra-low-power machine learning. It is based on logic operators, and its computational locality allows massively parallel computing. However, its application to machine learning has been limited due to a lack of training methodologies. In this paper, we extend a partitioned linear hybrid CA (P-LHCA) to a two-dimensional lattice to preserve the input image space geometry. We show that a 2D P-LHCA improves image classification accuracy by up to 17.8% on test sets, compared to a 1D P-LHCA with a similar parameter size. We also propose the first hardware design for a hybrid CA and benchmark it on the AMD Alveo V80 FPGA. Our prototype achieves over 300× speed-up compared to the GPU-based implementation while using less than 1 watt of power, demonstrating significant energy efficiency of non-arithmetic computation. The genetic algorithm remains a primary limitation of P-LHCA architecture, especially in 2D, because of its exponential growth in parameter size and slower training due to a higher degree of freedom in information propagation.**

***Index Terms*—cellular automata, machine learning, image classification, dynamical systems, combinatorial mathematics**

I. Introduction

Incorporating proximity into a computational model is inevitable for image classification. The closer the pixels are, the more likely they correlate [1]. Therefore, it is logical to consider the vicinity of pixels for any pixel-wise statistics. The first *trainable* image classifier that accounts for spatial locality is Fukushima's Cognitron, in which every pair of neurons is locally connected to unique but overlapping receptive fields [2]. Its successor, Neocognitron [3], led to a convolutional neural network (CNN) [4], which is the *de facto* standard for image classification due to its outstanding performance.

A cellular automaton (CA) is a computational model proposed by von Neumann to formulate biological systems, dating back to the 1940s [5]. Besides their simple constructions, some cellular automata [6], [7] are known to exhibit sophisticated, often unpredictable behavior. Banks first demonstrated that a two-dimensional CA can implement a universal machine [8]. In the early 1980s, Wolfram developed a systematic classification of cellular automata [9] based on his extensive study of an Elementary CA (ECA) [6]. Mitchell et al. first applied a genetic algorithm to cellular automata to solve the firing-squad problem in a one-dimensional homogeneous CA [10]. Their work laid the foundation for machine learning in cellular automata—optimizing the cells' operating rules for a specific problem rather than coding a solver in their initial conditions. In 1986, Pries et al. coined the term "hybrid CA" [11], referring to a CA in which cells have different ECA rules. Later in the 1990s, Chaudhuri et al. studied a 90/150 hybrid ECA [12], which led to a Generalized Multiple Attractor Cellular Automaton (G-MACA) [13], the first CA model that implements associative memory for *arbitrary* data. A G-MACA is inherently limited to a small lattice of dozens of cells, since local attractors must be mapped through numerical iterations. In 2024, we proposed a partitioned linear hybrid cellular automaton (P-LHCA) [14], the first CA-based machine learning architecture trainable on real-world data. A P-LHCA utilizes lattice partitioning and nonconvergent dynamics (i.e., Class 3 or 4 CA [9]) to scale beyond a thousand cells.

In this paper, we extend a P-LHCA to a two-dimensional lattice so that the geometry of input images is preserved. A cellular automaton (CA) has two advantages for energy-constrained applications. First, each processing element, called a cell, operates in parallel and autonomously. In comparison, a multilayer perceptron can be parallelized only within a single layer. Each hidden layer must know the output of its adjacent layers [4]. Second, a cell operation is fully described by a finite look-up table and does not require arithmetic operations [5]. Consequently, a CA can be realized on hardware using only combinational logic [11]. Cellular automata compute natively without floating-point units. They are known to be complex and power-hungry, increasing power budgets for machine learning inferences [15]. It might be noteworthy that a P-LHCA likely has biological plausibility. Its computational structure resembles that of the cortical column [14], [16], a unique anatomical structure that exists in the neocortex [17].

The remainder of the paper is organized as follows. In Section II, we propose our novel 2D extension for a P-LHCA. In Section III, we briefly introduce the lattice synthesis procedure. In Section IV, we describe our hardware accelerator design for a P-LHCA. In Section V, we first analyze the density of Boolean functions in a 1D and 2D P-LHCA with different topologies. We then train a 2D P-LHCA on the MNIST and K-MNIST datasets and compare

This work is funded by ONR grant N00014-21-1-2324.

its performance with that of its 1D counterpart. Lastly, we benchmark our FPGA-based P-LHCA implementation against a GPU-based implementation in inference speed and actual power consumption. Section VI summarizes our findings and discusses the current challenges of P-LHCA architecture.

II. Definitions

A. Linear Hybrid CA

A cellular automaton (CA) is a discrete dynamical system that consists of multiple processing elements, called cells, over a lattice space. Each cell takes its neighbors' state as an input and evolves its state based on a pre-defined look-up table (LUT). If all cells share the same setting, the CA is said to be homogeneous. For example, a famous Wolfram's Elementary CA [6] and Conway's Game of Life [7] are homogeneous. A hybrid CA is a class of heterogeneous CA in which the cells differ only in LUTs (i.e., local state transition functions) [12]. The operation of the ith cell ($i = 1, 2, ..., N$) is defined by a tuple:

$$\begin{cases} \Sigma & \text{a state set} \\ \eta_i & \text{a neighbor set} \\ \phi_i : \Sigma^{|\eta_i|} \rightarrow \Sigma & \text{a look-up table (LUT)} \end{cases} \tag{1}$$

For the rest of the paper, we use $\Sigma = \{0, 1, 2, ...K-1\}$ and update all cells synchronously at every time step t. We also assume the null boundary conditions.

Let $S(t) = [s_1, s_2, ..., s_N]$ where s_i is the ith cell's state (a.k.a., configuration). A linear CA [9] uses a subset of LUTs that holds the additivity over a Galois field of order K:

$$\forall_{t>0},\ \phi^t\{S_a(0) + S_b(0)\} = \phi^t\{S_a(0)\} + \phi^t\{S_b(0)\} \tag{2}$$

As the field defines addition and multiplication, each look-up table can be specified solely by postsynaptic weights $w_{i,j} \in \Sigma$ on every ith cell's neighbor:

$$\phi_i = (\sum_{j \in \eta_i} w_j s_j(t-1)) \mod K \tag{3}$$

In vector form,

$$\begin{aligned} S(t) &= \mathbf{A}S(t-1) \mod K \\ &= \mathbf{A}^t S(0) \mod K \end{aligned} \tag{4}$$

where $\mathbf{A}$ is a N-by-N weighted adjacency matrix, referred to as the characteristic matrix [12] of a linear CA.

From (4), we can compute a state of a linear hybrid CA from an arbitrary initial state with integer operators. However, importantly, a *local* state transition function (ϕ_i) still consists of look-up tables. [18], [19] provide comprehensive instructions on constructing an additive CA.

B. Lattice Partitioning

A CA is autonomous, so an input from the external world must be embedded as an initial condition: $S(0)$. As proposed in [14], we partition the lattice into three regions: receptors, motors, and processors, so that the state space is not restricted to the input space. Let ξ be an index set for receptors and o be another set for motors such that $\xi, o \subset \{1, 2, 3, ..., N\}$ and $|\xi| + |o| \leq N$. Note that $|\xi|$ must be equal to the dimensionality of the input space and that K must be larger than or equal to its dynamic range. For example, if an image is binary, then K must be at least 2. Similarly, the cardinality of o must be larger than or equal to the number of classes. If a cell is in neither set, then it is called a processor.

The lattice output is determined by the motor states at a time τ. Each label is encoded as a $1 \times |o|$ one-hot vector. Optionally, a temporal window can be applied to the motors:

$$v_i = \sum_{u=0}^{\omega-1} s_i(\tau - u) \mod K,\ \omega > 0 \tag{5}$$

The winner-take-all function is then applied to each motor state. The index of the motor cell whose state (v_i) has the largest value is considered as a predicted label:

$$y = \begin{cases} j & \text{if } \forall_{i \neq j \in o}\ v_j > v_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

In summary, a P-LHCA requires the following hyperparameters: $\mathbf{A}$, K, ξ, o, τ, and ω. The trainable parameters are the weights on each neighbor/local connection, including the feedback term $w_{i,i}$, encoded as non-zero elements in $\mathbf{A}$. Each input is set to the initial conditions of receptors. Every cell collectively evolves its internal states through interaction with its neighbors over time. At the τ time step, the last ω states of motor cells are read out and applied to the winner-take-all function, determining the output of the lattice.

C. 2D Extension

To employ a P-LHCA over a 2-dimensional space, a cell neighborhood and lattice partitioning must be defined in a 2D lattice space. Fig. 1 visualizes two known neighborhoods for a 2D CA: von Neumann and Moore [20]. We propose three lattice partitioning topologies based on the placement of motor cells: unilateral, medial, and bilateral (i.e., diagonal). In the unilateral topology, all motor cells are placed on one side of the square lattice. In contrast, the bilateral topology places motor cells across the edges, so each side has both the motor and receptor cells. Lastly, the medial topology places the motor cells in the center of the square lattice. Fig. 2 illustrates the proposed lattice partitioning for a 2D lattice.

Notice that (4) holds for any dimensionality of the lattice as long as cell coordinates can be mapped into a finite index set. A different neighborhood alters only the locations of zero elements of $\mathbf{A}$. Thus, $\mathbf{A}$ along with K fully defines a linear CA in a 2D lattice. Unless otherwise stated, we index a cell by column-major order.

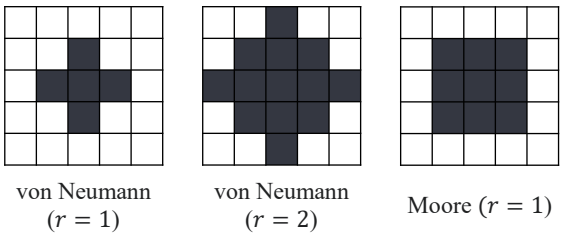

Fig. 1. 2D Cellular Automata Neighborhood

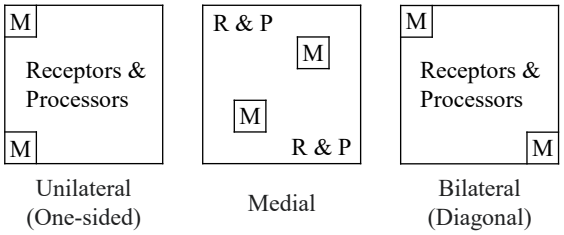

Fig. 2. Proposed 2D P-LHCA Topologies

III. CA Synthesis

The most important novelty of a partitioned linear hybrid CA (P-LHCA) [14] is that it is trainable. Synthesis [5] is the process of finding LUTs that best imitate input-output maps in arbitrary data pairs. Thus, its goal is to determine a set of LUTs that map receptor states to the desired motor states, thereby differentiating input classes at most. Assume M image label pairs are given. Synthesis seeks to maximize the fitness function:

$$F = \frac{1}{M}\sum_{j}^{M} f_j$$
$$f_j = \begin{cases} 1 & \text{if } y_j = d_j \text{ and } y_j \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

We employ a genetic algorithm [21] with elitism [22] as an optimizer. For the sake of space, we refer to [14] for a detailed procedure and provide here only the minimum description required to define the tunable parameters. We first randomly generate P chromosomes as an initial population. A chromosome comprises N genes that define each cell operation, which consists of $|\eta|$ nucleotides encoding each postsynaptic weight. Each chromosome is then realized as a P-LHCA and evaluated on training data. The chromosomes are ranked by their fitness values, and the top p_{Elite} portion of the population is copied to the next population pool without any modification. A rank-based selection [22] is applied to the population to form an intermediate population, in which the chromosomes are randomly selected for crossover and mutation, specified by the preset probabilities p_{Xross} and p_{Mut}, respectively. Lastly, the intermediate population is copied to the remainder of the next population. The evolution process continues until it reaches the preset maximum generation Q or the best fitness reaches 1.

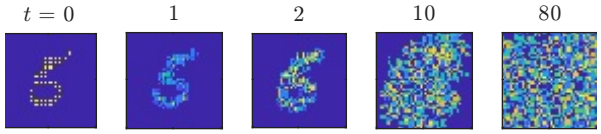

Fig. 3. Typical Space-Time Plot of a 2D P-LHCA. The input image ("5") is quickly distorted and encoded as phases, as the information propagates across the lattice over time. For better visualization, all processors are initially set to 0, and the Moore neighborhood of $r = 1$ is used.

IV. Hardware Implementation

To the best of our knowledge, a hardware accelerator for cellular automata is not publicly available. Therefore, we prototype our P-LHCA model on a Field-Programmable Gate Array (FPGA) to demonstrate energy efficiency of non-arithmetic computation. We target our hardware design for the AMD Alveo V80 Compute Accelerator[1], equipped with two 16 GB High Bandwidth Memory (HBM) stacks and the built-in PCI Express (PCIe) controller [23]. This hardware architecture is well-suited for cellular automata simulation as it minimizes I/O bottlenecks between the host computer and the FPGA fabric. Since a CA can evolve at each clock cycle, the computation takes much less time than conventional floating-point arithmetic-based algorithms.

Our developed P-LHCA accelerator consists of three major components: a host computer program (i.e., a card driver), a lattice circuit, and an onboard controller. Fig. 4 provides the top-level block diagram. All onboard components communicate over the AMBA AXI4 interface [24] with burst support.

The overall execution flow is as follows. First, the host computer program transfers input data and LUT configurations (i.e., trainable parameters) to the onboard HBM over the PCIe bus. Second, upon the *go* signal issued through the configuration registers, the onboard lattice reads an initial condition from the HBM, evolves a cellular automaton (i.e., performing computation), and writes the final lattice state into the HBM, until inputs are exhausted. Third, the onboard controller asserts the *done* signal[2]. Subsequently, the host program retrieves the lattice outputs from the HBM and decodes the cell states. On the host computer, the genetic algorithm is re-run and updates the parameters. The entire procedure is repeated until the learning criteria are met.

A. Host Computer Program

The host computer program plays two roles. Primarily, it configures the lattice circuit over the PCIe bus. The run-time hyperparameters include the lattice size (N), neighborhood size (r), readout time (τ), and batch size (M). They are written to the configuration registers exposed through the AXI4 Lite interface. In addition, the program encodes and decodes lattice

[1]The part number is `xcv80-lsva4737-2MHP-e-S`.

[2]Currently, the *done* signal must be constantly polled due to a lack of hardware interrupt support in the platform.

TABLE I
FREQUENCY OF PERFECT SOLUTIONS ($F = 1$) FOR ALL BOOLEAN FUNCTIONS

	1D ($r = 4$)			**2D**								
				von Neumann ($r = 1$)			**von Neumann** ($r = 2$)			**Moore** ($r = 1$)		
	Unilateral	Medial	Bilateral	Unilat.	Med.	Bil.	Unilat.	Med.	Bil.	Unilat.	Med.	Bil.
NAND	7.02	8.19	8.64	0.09	0.53	0.14	8.98	9.51	8.93	5.84	8.78	6.04
AND	7.19	8.06	8.58	0.09	0.51	0.13	8.93	9.49	8.95	5.89	8.81	5.94
NOR	7.28	8.19	8.65	0.11	0.48	0.13	8.99	9.54	8.95	5.90	8.81	6.10
OR	6.96	8.19	8.64	0.09	0.48	0.12	9.01	9.55	9.03	5.88	8.78	6.04
XNOR	9.35	8.91	9.84	0.30	1.01	0.42	10.50	11.07	10.51	8.45	10.52	8.30
XOR	9.41	8.89	9.88	0.30	0.95	0.38	10.60	11.15	10.61	8.32	10.68	8.46
Average	7.87	8.41	**9.04**	0.17	**0.66**	0.22	9.51	**10.05**	9.50	6.71	**9.40**	6.82

$\times 10^{-5}$

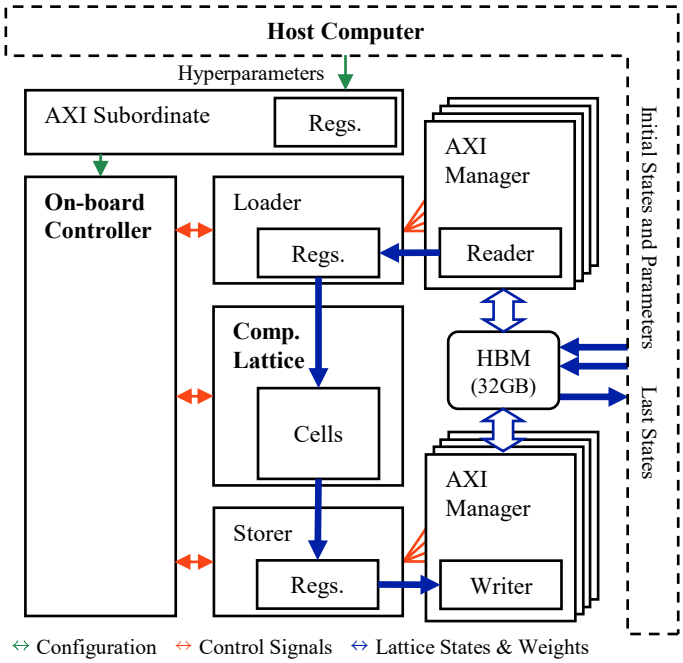

Fig. 4. Top-Level Block Diagram

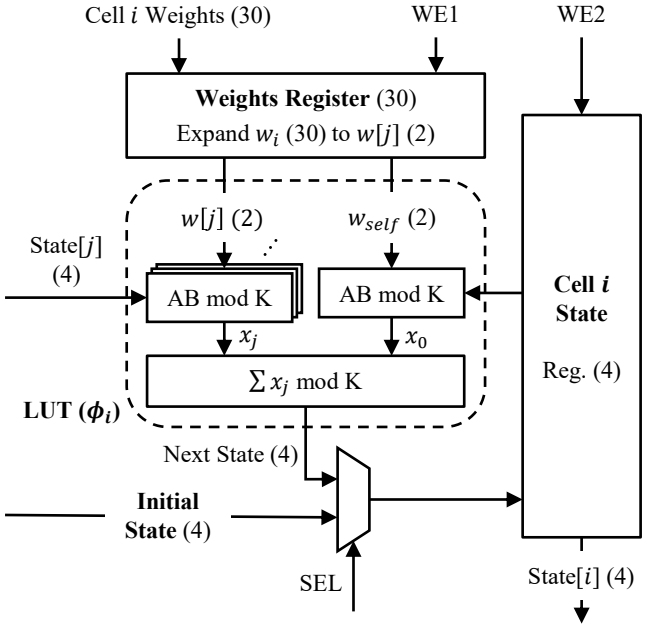

Fig. 5. Example of Reconfigurable Cell for an LHCA with $K = 16$ and $|\eta| = 15$. `SEL` is enabled only at $t = 0$. `WE1` is enabled during the initialization phase. `WE2` is enabled while the lattice is evolving. Note that all entities within the LUT block can be written as pure combinatorial logic. The binary operators used here are for drawing purposes only.

states and LUT configurations for a learning algorithm running on the host computer. Since AXI4 is byte-addressed, they must first be packed into a byte. For example, with three-valued logic, one byte constitutes four weights. The packed bytes are aligned per cell in little-endian and padded with zeros if their size is not a multiple of 4 bytes. The CPU-HBM communication is performed through the Xilinx PCIe Multi-Queue DMA (QDMA) Intellectual Property (IP) core [25].

B. Computational Lattice

The computational lattice is the critical part of our overall design. In essence, it takes initial conditions (i.e., inputs) and a set of logics (i.e., weights) and computes cell states (i.e, outputs) for arbitrary time steps. As shown in Fig. 5, all computation is combinatorial. A register is added to synchronize all cells at each clock, so one computation takes only a single clock. The maximum lattice size, maximum neighbor size, lattice shape (i.e., 1D vs 2D), and state set size (K) must be preset prior to synthesis (in FPGA terminology, not in the CA context), as these constraints determine the circuit size. Notice that a smaller lattice can be implemented at run-time by zeroing the weights of cells in an unused portion of the lattice. Similarly, different neighborhoods and/or neighbor sizes can be set on the fly unless they exceed the maximum size a synthesized circuit can support.

The lattice circuit has two auxiliary components: the loader and the storer that implement AXI4 managers. Using multiple AXI4 buses is preferred over a single-bus design to maximize the HBM-Lattice bandwidth, provided the V80's interconnect topology. In brief, each HBM stack consists of eight memory controllers, accessible through two pseudo channels that expose two AXI4 subordinates for each [26]. Since each

pseudo channel can access only a 1 GB segment of its parent HBM stack [26], it is optimal to split an input sample into 32, 1 GB chunks and to designate an AXI manager to each memory block. Currently, we utilize only two AXI4 managers per component. Because the lattice size is relatively small, additional channels provide little advantage while increasing hardware complexity and utilization, which can lower a clock speed without rigorous design optimization. For instance, a lattice of $N = 1024$ with $K = 16$ requires 512 bytes per input, which requires only 16 AXI transfers with 256-bit buses. Given that the memory latency is less than 25 clock cycles, each transaction takes only about 200 clock cycles with a two-controller design. In other words, if $\tau > 200$, the bottleneck is not the I/O but the lattice, which cannot run faster because it already computes its next output by one time step per clock.

C. On-board Controller

The onboard controller orchestrates all subcomponents to maximize overall bandwidth. It entirely separates the datapath from the control plane, as illustrated in Fig. 4. To give an example, it allows the loader and storer modules to run concurrently, doubling the HBM-Lattice bandwidth except for all but the first and last inputs. It also reduces the CPU-FPGA communication, which is the slowest link on the entire system.

V. Experimental Results

All P-LHCA simulations are performed on MathWorks MATLAB R2024b Update 3, except for Section V-C. Binary exponentiation is used to avoid overflow in computing the power of the characteristic matrix. In Section V-B, a convolutional neural network (CNN) is run on Python 3.13.5 with TensorFlow 2.20.0. The same Python version is used throughout the paper.

A. Boolean Operators

We first employ a Monte Carlo simulation to understand how different lattice topologies affect the mapping capabilities of a P-LHCA. Since we evaluate a P-LHCA for binary image classification, we estimate the number of possible solutions for all Boolean functions. With a higher average density of solutions, we assume that the lattice has more functional primitives to construct a desired mapper function.

For each Boolean function, 10^6 chromosomes are generated by randomly drawing a weight from a uniform distribution of $\{0, 1, 2\}$. We then compute their fitness values defined as (7). All 2D P-LHCAs are set to $K = 8$, $\tau = 50$, and $\omega = 1$. For a 1D lattice, we use $N = 39$ and $r = 4$. This configuration is picked so that no receptors and motors share direct connections, while the 1D and 2D automata have approximately the same number of cells. Note that a 2D lattice has a much larger parameter space than a 1D lattice for the same radius due to its larger neighborhood size, as shown in Fig. 1 and 6. For 2D lattices, $N = 36$ is used on a 6×6 rectangular grid, and different radii are tested based on their neighborhoods.

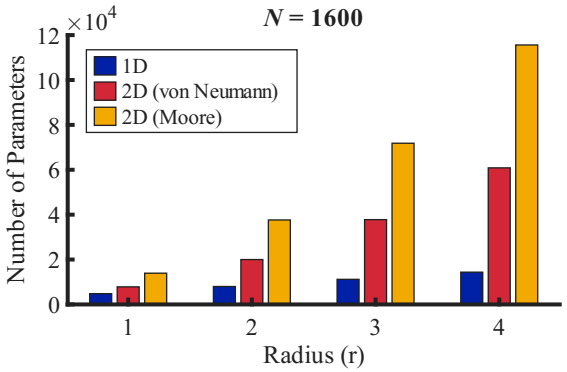

Fig. 6. Parameter Size of 1D and 2D P-LHCAs with Different Neighborhoods. The parameter space grows exponentially on a 2D lattice ($\approx r^2$) while constantly on a 1D lattice ($2r$). The fractions are due to the null boundary conditions.

The frequency of unique solutions is summarized in Table I. Each solution count is normalized by the number of possible parameters. Table I provides two notable insights. First, the topology defines the density of Boolean functions. For 1D, the bilateral topology has the highest density, so it is likely the most optimal topology. This observation aligns with the results reported in [14]—the bilateral topology achieves the highest classification accuracy among three different topologies. For 2D, the medial topology appears the most optimal.

Second, the 2D P-LHCAs have more solutions than any 1D models, implying 2D lattices provide a greater degree of freedom in learning the mappings. The poor performance with the von Neumann neighborhood of $r = 1$ is explainable. Since the activation cannot propagate diagonally, the lattice is equivalent to a 1D lattice of $N = 24$ with the periodic boundary conditions, except for the medial topology, where two motor cells are "tapped" onto the pathways between the receptors and processors at the peripheral.

B. Image Classification

We evaluate a 2D P-LHCA on two real-world datasets: MNIST [27] and K-MNIST [28], from which two subsets are prepared. Each subset contains only two classes but includes all samples (6000 training and 1000 test images) per class, so the dataset size is doubled compared to the previously reported results [14]. We select two pairs of classes for an easy dataset and a harder one based on visual similarity between classes. All images are binarized by the threshold of 120 and down-sampled to 16×16. The reduced image size and the number of classes (i.e., the motors) are not limitations of the P-LHCA architecture. However, we restrict the lattice size to be relatively small because the GA becomes notably slow for a lattice beyond $N = 2400$ [14].

To verify our pre-processed datasets, we trained a CNN with the ADAM optimizer [29] on the same four datasets as a baseline. The network topology provided in Fig. 7 is selected so that the total parameter size in single precision is close to that of a 2D-LHCA (approximately 3 KB for both models). It is important to recognize that the parameter size does not

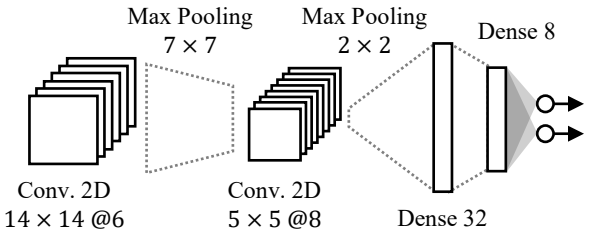

Fig. 7. CNN Topology. A small CNN with ReLU activation function is employed to verify the datasets. The learning rate is set to 0.001. The ADAM optimizer is configured with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$.

TABLE II
TRAINING ACCURACY

	Classes	P-LHCA		CNN
		1D	2D	
MNIST	0, 1	87.6±2.37	94.1±2.0	99.8
	5, 6	68.3±0.73	81.3±2.1	98.4
K-MNIST	0, 1	88.8±1.28	84.8±2.8	98.6
	5, 6	75.3±0.76	76.7±1.6	97.5

compare computational complexity or network size. However, it is a reasonable approximation to quantify the scale of two very different models.

We use the same lattice size for both 1D and 2D automata: $N = 1600$, $K = 16$, and $\omega = 3$, but with different topologies and neighbor sizes to match their parameter sizes as closely as possible. For a 1D P-LHCA, the bilateral topology with $r = 4$ is used. For a 2D P-LHCA, the medial topology but $r = 1$ with Moore neighborhood is chosen so that its neighbor size is close to that of the 1D lattice, resulting in 14380 weights for 1D and 13924 for 2D. Additionally, the readout times (τ) are adjusted as the activation propagates much faster on the 2D lattice. For $N = 1600$, the maximum travel distance on the 1D lattice is $\lceil N/r \rceil = 400$, while that of the 2D variant is $\lceil \sqrt{N}/r \rceil = 40$. We use $\tau = 500$ for 1D and $\tau = 80$ for 2D. The GA parameters are $P = 150$, $p_{\text{Elite}} = p_{\text{Xross}} = 0.1$, $p_{\text{Mut}} = 10^{-3}$, and $Q = 2000$. Ten different initializations are tested. A 5-fold cross-validation is used to pick the best chromosome for each dataset. The only chromosome with the highest validation accuracy is then evaluated on the test set.

The training fitness and the test classification accuracy

TABLE III
TEST ACCURACY

	Classes	P-LHCA		CNN
		1D	2D	
MNIST	0, 1	88.2	93.9	99.8
	5, 6	63.4	81.2	98.8
K-MNIST	0, 1	88.2	79.5	97.0
	5, 6	56.5	61.5	94.3

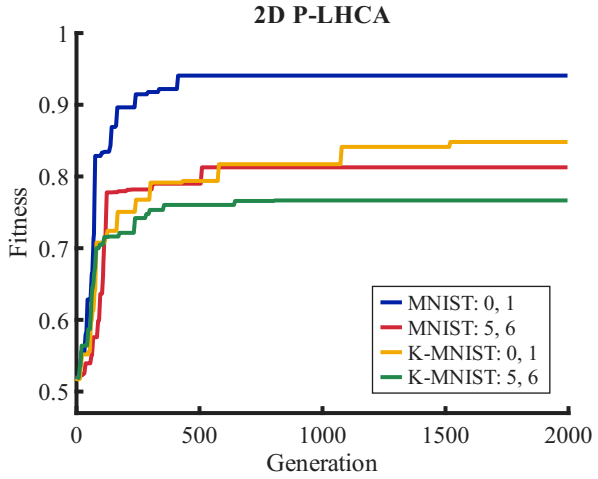

Fig. 8. Learning Curves of 2D P-LHCA. For all datasets, the training converges.

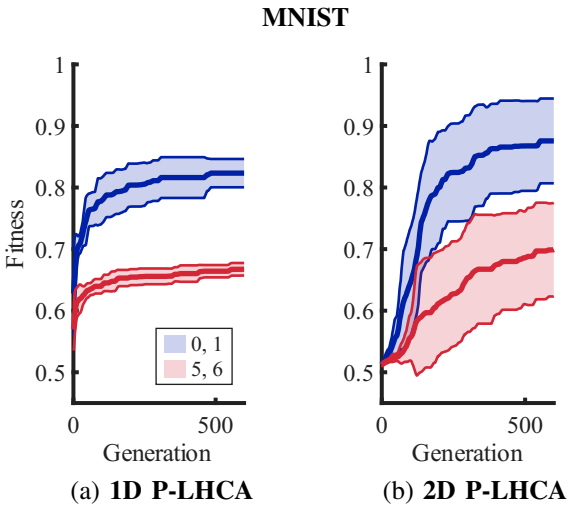

Fig. 9. Variance among Training Sessions on MNIST Dataset. The shaded area is the standard deviation of fitness across 10 trials. Noticeably, the 2D P-LHCA has a much higher variance, while the 1D model has slightly more parameters.

are provided in Table II and III. The 2D P-LHCA overall outperforms the 1D P-LHCA, particularly on tougher datasets where two digits/characters resemble one another. However, Fig. 8 and Fig. 9 show that the 2D P-LHCA has slower training and higher trial variances than the 1D P-LHCA, which might explain its inferior performance on the K-MNIST dataset of the classes 0 and 1. We conjecture that this higher training variance is due to the increased degree of freedom in the propagation of information in the lattice. In 1D, a cell activation (that is, input information) can propagate only in two directions (i.e., to the left or right). By contrast, the information can travel in any of the eight directions (i.e., north, west, northeast, etc.) in 2D. Thus, the number of possible "paths" from receptors to motors explodes within much fewer time steps. We leave theoretical proof for future work.

TABLE IV
FPGA PRIMITIVES UTILIZATION AND ESTIMATED POWER CONSUMPTION

Lattice Size	256	512	1024
LUT	16191	31550	62807
FF	15029	29369	58064
BRAM	0	0	0
DSP	0	0	0
Clock (MHz)	200	200	200
Power (W)	0.258	0.489	0.921

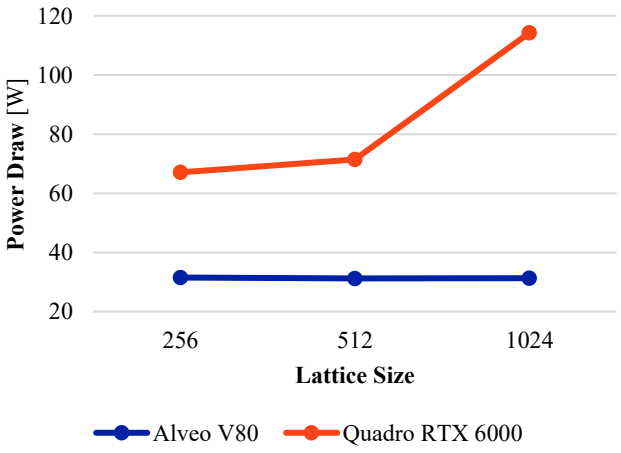

Fig. 10. Actual Power Usage during Inference

C. Hardware Utilization

Our designed circuit is successfully synthesized by the AMD Vivado ML 2023.2 software and tested on the AMD Alveo V80 accelerator board [23]. The developed accelerator IP core is integrated with the AMD Alveo Versal Example Design (AVED) platform[3] [30] and connected to `HBM_NSU` ports on the Network-on-Chip. The QDMA IP is connected to the PCIe 3.0 ×8 link provided by the Intel Xeon Gold 6244 CPU. The host computer software runs on Python 3.13.5, along with the Xilinx QDMA driver version 2024.1, on the Ubuntu 22.04.5 LTS operating system. The vendor-provided `ami_tool` utility is used to poll the power usage from the Real-time Processing Unit subsystem on the FPGA board. For a baseline comparison, the inference code is re-implemented in Python with CuPy 13.6.0 and run on a single NVIDIA Quadro 6000 RTX GPU with CUDA 13.8. The `nvidia-smi` tool is used to measure the GPU power consumption. For both FPGA and GPU power measurements, the polling interval is set to 1 second.

Table IV reports the post-implementation resource utilization. Notably, the lattice IP consumes very little power, less than 1 watt. In addition, the computational lattice does not require any DSP slices as all computation is combinational. As expected, larger lattices require more primitives and raise the power budget. That said, Fig. 10 shows that the measured power increase is marginal in contrast to that of the GPU implementation. We speculate that GPU power usage increases as a larger lattice results in a larger characteristic matrix, leading to more multiplications. By contrast, the power increase in our hardware prototype is much smaller because the power consumption of the computational lattice is far less compared to the other onboard hardware components (e.g., a Processing System).

[3]We use the release version `amd_v80_gen5x8_exdes_2_20240408`.

TABLE V
FPGA VERSUS GPU INFERENCE SPEED COMPARISON

$N_{\max}$	FPGA (Proc. Only)	FPGA	GPU	Speed-up
256	1610.3	866.9	2.7	320.9×
512	1589.0	832.4	2.8	299.2×
1024	1663.8	824.9	2.6	322.2×

[chromosome / second]

The lattices have the 2D topology with the Moore neighborhood. The runtime configurations are $M = 15000$, $\tau = 200$, and $r = 1$. As a reminder, a chromosome is a set of weights that fully defines a P-LHCA, which is updated by the GA. For measuring processing-only time, we take the duration between the *go* and *done* signals are first asserted.

TABLE VI
SPEED GAIN OVER BATCH SIZE AND READOUT TIME ($N = 1024$)

Readout (τ)	# of Inputs (M)		
	10000	**20000**	**30000**
100	288.1×	497.2×	515.8×
200	241.0×	352.3×	403.4×
300	206.5×	300.4×	287.5×

Finally, Table V and Table VI summarize the inference speeds achieved by our hardware implementation. We observe two factors determining the overall inference speed gain: batch size (i.e., the number of input samples loaded to the HBM at a time) and the readout time (i.e., how many times the lattice is evolved per input). It is noticeable from Table V that the CPU-HBM data transfer is the current bottleneck of our hardware design. This is not a surprise since the data has to travel over limited PCIe bandwidth and through multiple software stacks on the host computer. That being said, we doubt the CPU-HBM bandwidth is problematic in real-world applications. The onboard HBM is fairly large (32 GB), and the onboard controller can select which samples to evaluate. An ultimate solution would be running a learning algorithm on a computational lattice, enabling "learning-on-chip." Regardless, for the scope of this paper, an FPGA-based genetic algorithm adds significant hardware complexity, while we believe there is a more optimal optimizer for hybrid CA, as discussed in Section VI.

Table VI shows that a larger batch size favors the GPU implementation. This is also expected, particularly for a *linear*

CA, because (4) allows computing the final output without iterating over every time step. For the same reason, a longer readout time reduces the speed-up as the lattice must be evolved more times on the computational lattice. In contrast, on the GPU, such an increase in computational cost is only one-time for computing the characteristic matrix. Still, the developed accelerator provides a drastic speed-up over the GPU-based implementation, up to $515.8\times$, for using less than 1 *watt* of power.

VI. Conclusion

We proposed new P-LHCA topologies extending its application to the two-dimensional lattice. The Monte Carlo simulation showed that a 2D P-LHCA realizes more Boolean functions than its 1D counterpart, and that the medial topology is likely superior to the other 2D topologies. It is also demonstrated that a 2D P-LHCA improves the classification performance up to 17.8%, while the 2D variant has fewer parameters than the 1D model. A 2D P-LHCA preserves the spatial structures of input images, which may contribute to performance gains. If two pixels are close together, they interact on the lattice sooner than those farther apart. The cost of adapting a 2D P-LHCA is exponential in the parameter space per radius. It is also apparent that a 2D P-LHCA has much slower training with higher variance, which we believe is caused by an additional degree of freedom in information propagation.

Additionally, we implemented and benchmarked our novel hybrid CA hardware on the AMD Alveo V80 FPGA, achieving up to $515.8\times$ speed-up in inference compared to the GPU-based implementation. To the authors' knowledge, this is the first publicly reported hybrid CA accelerator. Our hardware implementation consumes only a hundredth of the GPU's power, demonstrating the potential for significant energy efficiency in a CA-driven machine learning system. That said, learning scalability must be improved for real-world applications. In particular, a faster optimization algorithm that exploits the locality in lattice space is required to synthesize a larger lattice. A P-LHCA stores much less information per processing element than perceptrons because it has very sparse intercellular connections with discrete parameters. Despite these challenges for future work, we believe this paper provides a stepping stone for non-arithmetic machine learning, potentially reducing energy costs for future artificial intelligence platforms.

Acknowledgment

The authors thank AMD University Program for donating a Vivado ML license, and Raul Valle and Benjamin L. Colburn for proofreading.

References

[1] W. R. Tobler, "A computer movie simulating urban growth in the detroit region," *Econ. Geogr.*, vol. 46, pp. 234–240, Jun. 1970, doi: 10.2307/143141.

[2] K. Fukushima, "Cognitron: a self-organizing multilayered neural network," *Biol. Cybern.*, vol. 20, no. 3, pp. 121–136, Sep. 1975, doi: 10.1007/BF00342633.

[3] ——, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nat*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[5] J. von Neumann and A. W. Burks, *Theory of Self-Reproducing Automata*. USA: University of Illinois Press, 1966.

[6] S. Wolfram, "Statistical mechanics of cellular automata," *Rev. Mod. Phys.*, vol. 55, pp. 601–644, Jul. 1983, doi: 10.1103/RevModPhys.55.601.

[7] M. Gardner, "Mathematical Games," *Sci. Am.*, vol. 223, no. 4, pp. 120–123, Oct. 1970, doi: 10.1038/scientificamerican1070-120.

[8] E. R. Banks, "Universality in cellular automata," in *11th Annu. Symp. Switch. Automata Theory*, Oct. 1970, pp. 194–215, doi: 10.1109/SWAT.1970.27.

[9] S. Wolfram, "Universality and complexity in cellular automata," *Physica D*, vol. 10, no. 1, pp. 1–35, Jun. 1984, doi: 10.1016/0167-2789(84)90245-8.

[10] M. Mitchell, J. P. Crutchfield, and R. Das, "Evolving cellular automata with genetic algorithms: A review of recent work," in *Proc. 1st Int. Conf. Evol. Comput. and Appl.*, Moscow, Russia, Jun. 1996. [Online]. Available: https://melaniemitchell.me/PapersContent/evca-review.pdf

[11] W. Pries, A. Thanailakis, and H. C. Card, "Group properties of cellular automata and vlsi applications," *IEEE Trans. Comput.*, vol. C-35, no. 12, pp. 1013–1024, Dec. 1986, doi: 10.1109/TC.1986.1676709.

[12] S. Nandi and P. P. Chaudhuri, "Analysis of periodic and intermediate boundary 90/150 cellular automata," *IEEE Trans. Comput.*, vol. 45, no. 1, pp. 1–12, Jan. 1996, doi: 10.1109/12.481481.

[13] N. Ganguly, P. Maji, B. K. Sikdar, and P. P. Chaudhuri, "Generalized multiple attractor cellular automata (GMACA) model for associative memory," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 16, no. 7, pp. 781–795, Nov. 2002, doi: 10.1142/S0218001402001988.

[14] N. Sawahashi and J. C. Principe, "End-to-end image classification in linear hybrid cellular automata," in *Int. Joint Conf. on Neural Netw.*, Jun. 2024, doi: 10.1109/IJCNN60899.2024.10650865.

[15] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint*, Nov. 2018, doi: 10.48550/arXiv.1811.01721.

[16] Y. Burnod, *An Adaptive Neural Network: The Cerebral Cortex*. Prentice-Hall, 1990.

[17] V. B. Mountcastle, "Modality and topographic properties of single neurons of cat's somatic sensory cortex," *J. Neurophysiol.*, vol. 20, no. 4, pp. 408–434, Jul. 1957, doi: 10.1152/jn.1957.20.4.408.

[18] O. Martin, A. M. Odlyzko, and S. Wolfram, "Algebraic properties of cellular automata," *Commun. Math. Phys.*, vol. 93, no. 2, pp. 219–258, Jun. 1984, doi: 10.1007/BF01223745.

[19] P. P. Chaudhuri, D. R. Chawdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata Theory and Applications Volume I*. IEEE Comput. Soc. Press, 1997.

[20] M. Mitchell, "Computation in cellular automata: A selected review." *Non-standard computation*, pp. 95–140, 1998.

[21] J. H. Holland, "Genetic algorithms," *Sci. Am.*, vol. 267, no. 1, pp. 66–73, Jul. 1992, doi: 10.1038/scientificamerican0792-66.

[22] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu, *Evolutionary computation*. CRC Press, 2000.

[23] Advanced Micro Devices Inc., *Alveo V80 Data Center Accelerator Cards Data Sheet (DS1013)*, May 2025.

[24] Arm Limited, *AMBA AXI and ACE Protocol Specification*, Mar. 2020.

[25] Advanced Micro Devices Inc., *QDMA Subsystem for PCI Express Product Guide (PG302)*, May 2024.

[26] ——, *Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller 1.1 LogiCORE IP Product Guide (PG313)*, Dec. 2025.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[28] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical Japanese literature," *arXiv preprint*, Dec. 2018, doi: 10.48550/arXiv.1812.01718.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint*, Dec. 2014, doi: 10.48550/arXiv.1412.6980.

[30] Advanced Micro Devices Inc., *Alveo Versal Example Design (AVED)*, May 2024. [Online]. Available: https://xilinx.github.io/AVED/amd_v80_gen5x8_exdes_2_20240408/index.html