JOURNAL OF FUNDAMENTAL MATHEMATICS
AND APPLICATIONS (JFMA) VOL. 9 NO. 1 (2026)

batara-guru: A Python-Based Rule 30 Cellular Automaton Analyzer with Parallel Processing

Sandy H. S. Herho[1*], Gandhi Napitupulu[2]

[1] *Department of Earth and Planetary Sciences, University of California, Riverside, CA, USA 92521.*
[2] *Coastal Hazards and Energy System Science (CHESS) Lab, Hiroshima University, Hiroshima, Hiroshima Prefecture, Japan 739-8529.*
Email: [1]sandy.herho@email.ucr.edu, [2]d243631@hiroshima-u.ac.jp
*Corresponding author

Abstract. We present `batara-guru`, an open-source Python library for efficient Rule 30 cellular automaton simulation and analysis. Using Numba's just-in-time compilation and parallel processing, the implementation achieves sub-second execution for systems up to 2001 cells on standard laptop hardware, overcoming Python's traditional performance limitations. The library integrates rigorous mathematical formulation from discrete dynamical systems theory with practical tools including NetCDF data persistence and diagnostic metrics for compositional/local entropy and interface density. Performance benchmarks demonstrate simulation times of 0.01–1.27 seconds across spatial scales from 251 to 2001 cells. Analysis reveals scale-invariant properties with entropy converging to 0.73 bits and interface density saturating near 0.51, confirming Rule 30's chaotic behavior while indicating subtle deviations from randomness. By requiring only standard Python libraries and modest computational resources, this framework democratizes cellular automaton research, enabling sophisticated dynamical systems analysis without specialized hardware infrastructure.

Keywords: Cellular automata, Compositional entropy, Interface density, Parallel computing, Rule 30.

I. INTRODUCTION

Cellular automata constitute a class of discrete dynamical systems wherein local interaction rules generate emergent spatiotemporal complexity [1, 2]. These systems, first conceived by von Neumann in the context of self-replication [3], have since found applications across diverse fields including statistical mechanics [4], cryptography [5], pattern formation [6], and theoretical computer science [7]. The appeal of cellular automata lies in their computational simplicity—evolution proceeds through synchronous application of deterministic local rules—coupled with their capacity to exhibit behaviors ranging from stable equilibria to chaotic dynamics.

Among elementary cellular automata, Rule 30 occupies a distinguished position due to its complex, apparently random behavior despite deterministic evolution from simple initial conditions [1, 8]. Wolfram classified Rule 30 as Class III in his taxonomy of cellular automaton behaviors, characterized by chaotic dynamics, sensitivity to initial conditions, and absence of obvious regularities [2]. The rule's apparent randomness led to its adoption as a pseudorandom number generator in Mathematica [8], while its one-dimensional simplicity makes it amenable to rigorous mathematical analysis that remains elusive for higher-dimensional systems [9].

Theoretical investigations have established several properties of Rule 30. The system ex-

hibits sensitive dependence on initial conditions reminiscent of continuous chaotic systems, though the discrete state space precludes direct application of Lyapunov exponent analysis [10]. Studies of ensemble dynamics reveal relaxation toward high-entropy empirical measures under certain measures [11], suggesting connections to statistical mechanics despite the deterministic nature of information propagation at the microscopic level. The mathematical structure underlying this complexity remains incompletely understood—unlike linear rules such as Rule 90, which admit complete characterization via algebraic methods [12], Rule 30's nonlinearity resists closed-form analysis.

Recent developments have further elevated interest in cellular automata as fundamental computational structures. The Wolfram Physics Project proposes that spacetime and quantum mechanics emerge from simple computational rules operating on hypergraphs, with cellular automata serving as pedagogical models for understanding these more complex systems [13, 14]. This ambitious framework suggests that the apparent randomness and emergent complexity observed in Rule 30 may reflect fundamental aspects of physical reality rather than mere mathematical curiosity. The project's emphasis on computational irreducibility—the notion that certain systems require explicit simulation rather than analytical shortcuts—reinforces the importance of efficient numerical implementations for exploring these models [13]. While the physics interpretation remains speculative, it underscores cellular automata's continued relevance as minimal models for investigating how simple rules generate complex phenomena.

Computational approaches to studying cellular automata have evolved considerably since Wolfram's initial explorations on early microcomputers [1]. Modern investigations leverage parallel architectures to explore larger spatiotemporal domains and ensemble statistics [15]. Graphics processing units have enabled massive parallelization for certain cellular automaton classes [16], though their application to one-dimensional systems offers limited advantages given communication overhead. For elementary rules, CPU-based implementations with SIMD vectorization and multi-threading provide efficient execution on contemporary workstations [17].

Despite this progress, accessible tools for rigorous computational study of elementary cellular automata remain limited. Proprietary software environments impose licensing barriers and restrict algorithmic transparency, while specialized simulation codes often lack standardized data formats or comprehensive documentation. This gap motivates development of open-source frameworks that combine mathematical rigor with computational efficiency and methodological reproducibility [18, 19, 20].

The present work introduces `batara-guru`, a Python-based library for simulation and analysis of Rule 30 cellular automata with emphasis on parallel processing and scientific data management. Our implementation provides several contributions: rigorous mathematical formulation connecting discrete dynamical systems theory to computational practice; efficient numerical kernels achieving performance comparable to compiled languages through just-in-time (JIT) compilation; standardized data persistence via NetCDF format enabling interoperability with climate science analysis tools; comprehensive computation of diagnostic quantities including compositional entropy and interface density; and open-source distribution under permissive licensing facilitating community extension and validation. We focus specifically on Rule 30 rather than implementing a general cellular automaton framework to enable aggressive optimization exploiting its algebraic structure while providing students and researchers with a tractable entry point to nonlinear dynamics and computational physics.

II. METHODS

2.1. Model Formulation

We establish the mathematical foundations of Rule 30 cellular automata through the lens of discrete dynamical systems theory. Our treatment follows Hedlund's seminal work on shift dynamical systems [21], adapted to the finite periodic case, with particular attention to the algebraic and topological structures that govern evolution.

Definition 1 (State Space) *Fix $N \in \mathbb{N}$ with $N \geq 3$. The state space is the set of all binary configurations $\mathcal{C}_N = \{0,1\}^N$, viewed as the $\mathbb{Z}_2$-vector space with componentwise addition modulo 2 (XOR). The cyclic identification $\mathbb{Z}_N \cong \mathbb{Z}/N\mathbb{Z}$ gives $s_{i+N} = s_i$ for all $i \in \mathbb{Z}$.*

The cyclic indexing imposes toroidal topology on the discrete spatial lattice, eliminating boundary artifacts. This construction yields $|\mathcal{C}_N| = 2^N$ distinct configurations.

Definition 2 (Neighborhood Structure) *For $i \in \mathbb{Z}_N$ and radius $r \in \mathbb{N}$, the r-neighborhood of site i is the ordered tuple*

$$\mathcal{N}_r(i) = (i-r, i-r+1, \ldots, i, \ldots, i+r-1, i+r) \in \mathbb{Z}_N^{2r+1}.$$

The neighborhood map $\pi_i^{(r)} : \mathcal{C}_N \to \mathbb{Z}_2^{2r+1}$ extracts the local configuration via

$$\pi_i^{(r)}(s) = (s_{i-r}, s_{i-r+1}, \ldots, s_{i+r}).$$

Elementary cellular automata employ $r = 1$, giving a three-cell window at each site. Wolfram's classification scheme [1] enumerates all $2^8 = 256$ possible local rules by their action on the eight possible neighborhoods $\mathbb{Z}_2^3$.

Definition 3 (Local Rule) [1] *A local rule of radius r is any function $\phi : \mathbb{Z}_2^{2r+1} \to \mathbb{Z}_2$. For elementary rules with $r = 1$, we identify ϕ with its truth table $(b_7, b_6, \ldots, b_1, b_0) \in \mathbb{Z}_2^8$, where*

$$\phi(s_{-1}, s_0, s_1) = b_k \quad \textit{when} \quad k = 4s_{-1} + 2s_0 + s_1.$$

The Wolfram code of ϕ is the integer $w(\phi) = \sum_{k=0}^{7} b_k \cdot 2^k \in \{0, 1, \ldots, 255\}$.

This binary encoding provides a bijection between elementary rules and bytes. Rule 30 corresponds to the binary expansion $30 = 16 + 8 + 4 + 2 = 2^4 + 2^3 + 2^2 + 2^1$, yielding the bit pattern $(0, 0, 0, 1, 1, 1, 1, 0)$.

Definition 4 (Rule 30) [1] *The Rule 30 local function $\phi_{30} : \mathbb{Z}_2^3 \to \mathbb{Z}_2$ is specified by its truth table:*

(s_{-1}, s_0, s_1)	$(1,1,1)$	$(1,1,0)$	$(1,0,1)$	$(1,0,0)$	$(0,1,1)$	$(0,1,0)$	$(0,0,1)$	$(0,0,0)$
$\phi_{30}(s_{-1}, s_0, s_1)$	0	0	0	1	1	1	1	0

Direct inspection reveals the algebraic form

$$\phi_{30}(s_{-1}, s_0, s_1) = s_{-1} \oplus (s_0 \vee s_1), \tag{1}$$

where $\oplus$ and $\vee$ denote XOR and OR operations in the Boolean algebra $\mathbb{Z}_2$.

The verification of equation (1) proceeds by exhaustive checking against the truth table. Note that this representation is not unique—alternative Boolean polynomials yield the same input-output behavior.

Definition 5 (Global Dynamics) [21] *Given a local rule $\phi : \mathbb{Z}_2^{2r+1} \to \mathbb{Z}_2$, the induced global map $F_\phi : \mathcal{C}_N \to \mathcal{C}_N$ acts componentwise according to*

$$(F_\phi(s))_i = \phi(\pi_i^{(r)}(s)) = \phi(s_{i-r}, \ldots, s_{i+r}), \quad i \in \mathbb{Z}_N. \tag{2}$$

We write $F_{30} := F_{\phi_{30}}$ for the Rule 30 global evolution operator.

The map F_ϕ is well-defined since each output coordinate depends only on finitely many input coordinates. The periodic boundary ensures that no special treatment is needed at the edges of the lattice.

Lemma 1 (Shift-Invariance) *Let $\sigma : \mathcal{C}_N \to \mathcal{C}_N$ denote the cyclic shift operator defined by $(\sigma(s))_i = s_{i+1}$. Then for any translation-invariant local rule ϕ, we have $F_\phi \circ \sigma = \sigma \circ F_\phi$.*

Proof. For any $s \in \mathcal{C}_N$ and $i \in \mathbb{Z}_N$, we compute

$$\begin{aligned}(F_\phi(\sigma(s)))_i &= \phi((\sigma(s))_{i-r}, \ldots, (\sigma(s))_{i+r}) &(3)\\ &= \phi(s_{i-r+1}, \ldots, s_{i+r+1}) &(4)\\ &= (F_\phi(s))_{i+1} = (\sigma(F_\phi(s)))_i, &(5)\end{aligned}$$

where the third equality uses translation invariance of ϕ. Since i was arbitrary, $F_\phi \circ \sigma = \sigma \circ F_\phi$ as claimed. □

This commutativity reflects the spatial homogeneity of the cellular automaton—the update rule applies uniformly at all lattice sites.

Theorem 1 (Nonlinearity of Rule 30) *The global map $F_{30} : \mathbb{Z}_2^N \to \mathbb{Z}_2^N$ is not $\mathbb{Z}_2$-linear for any $N \geq 2$. That is, there exist configurations $s, s' \in \mathbb{Z}_2^N$ such that $F_{30}(s + s') \neq F_{30}(s) + F_{30}(s')$, where addition is componentwise modulo 2.*

Proof. We exhibit an explicit counterexample. Take $N = 3$ and consider

$$s = (1, 0, 0), \quad s' = (0, 1, 0).$$

Then $s + s' = (1, 1, 0)$ by componentwise addition in $\mathbb{Z}_2^3$.

Computing $F_{30}(s)$ coordinate by coordinate using equation (1):

$$\begin{aligned}(F_{30}(s))_0 &= \phi_{30}(s_2, s_0, s_1) = \phi_{30}(0, 1, 0) = 0 \oplus (1 \vee 0) = 0 \oplus 1 = 1, &(6)\\ (F_{30}(s))_1 &= \phi_{30}(s_0, s_1, s_2) = \phi_{30}(1, 0, 0) = 1 \oplus (0 \vee 0) = 1, &(7)\\ (F_{30}(s))_2 &= \phi_{30}(s_1, s_2, s_0) = \phi_{30}(0, 0, 1) = 0 \oplus (0 \vee 1) = 1. &(8)\end{aligned}$$

Thus $F_{30}(s) = (1, 1, 1)$.

Similarly for s':

$$(F_{30}(s'))_0 = \phi_{30}(0, 0, 1) = 1, \tag{9}$$
$$(F_{30}(s'))_1 = \phi_{30}(0, 1, 0) = 1, \tag{10}$$
$$(F_{30}(s'))_2 = \phi_{30}(1, 0, 0) = 1. \tag{11}$$

So $F_{30}(s') = (1, 1, 1)$ as well.

Now compute $F_{30}(s + s')$ where $s + s' = (1, 1, 0)$:

$$(F_{30}(s + s'))_0 = \phi_{30}(0, 1, 1) = 0 \oplus (1 \vee 1) = 1, \tag{12}$$
$$(F_{30}(s + s'))_1 = \phi_{30}(1, 1, 0) = 1 \oplus (1 \vee 0) = 0, \tag{13}$$
$$(F_{30}(s + s'))_2 = \phi_{30}(1, 0, 1) = 1 \oplus (0 \vee 1) = 0. \tag{14}$$

Therefore $F_{30}(s + s') = (1, 0, 0)$.

But $F_{30}(s) + F_{30}(s') = (1, 1, 1) + (1, 1, 1) = (0, 0, 0)$ in $\mathbb{Z}_2^3$. Since

$$(1, 0, 0) \neq (0, 0, 0),$$

we conclude that F_{30} violates additivity and hence is nonlinear. □

The nonlinearity stems fundamentally from the OR operation in equation (1), which fails to distribute over addition in $\mathbb{Z}_2$. In contrast, Rule 90 defined by $\phi_{90}(s_{-1}, s_0, s_1) = s_{-1} \oplus s_1$ is $\mathbb{Z}_2$-linear and hence analytically tractable [1].

Definition 6 (Orbit) [22] *For initial configuration $s^{(0)} \in \mathcal{C}_N$, the forward orbit or trajectory under F_{30} is the sequence*

$$\mathcal{O}(s^{(0)}) = \{s^{(t)}\}_{t=0}^{\infty}, \quad \textit{where} \quad s^{(t+1)} = F_{30}(s^{(t)}).$$

Since $\mathcal{C}_N$ is finite, every orbit is eventually periodic.

The orbit structure partitions $\mathcal{C}_N$ into basins of attraction for periodic cycles. For Rule 30, numerical evidence suggests highly intricate basin geometries [1], though rigorous analysis remains incomplete.

We now introduce quantitative measures to characterize configuration complexity. Our approach distinguishes between compositional mixing (measured via Shannon entropy) and spatial heterogeneity (measured via interface density).

Definition 7 (Density Functional) *The density functional $\rho : \mathcal{C}_N \to [0, 1]$ assigns to each configuration the proportion of active cells:*

$$\rho(s) = \frac{1}{N} \sum_{i=0}^{N-1} s_i. \tag{15}$$

Observe that ρ factors through the Hamming weight: $\rho(s) = \frac{1}{N}\|s\|_0$ where $\|s\|_0 = |\{i :$

$s_i = 1\}|$ counts the nonzero entries.

Definition 8 (Binary Entropy Function) [23] *The binary entropy function* $h : [0,1] \to [0,1]$ *is defined by*

$$h(p) = \begin{cases} -p\log_2 p - (1-p)\log_2(1-p) & \text{if } p \in (0,1), \\ 0 & \text{if } p \in \{0,1\}, \end{cases} \tag{16}$$

with the understanding that $\lim_{x\to 0^+} x\log_2 x = 0$.

This function is strictly concave on $[0,1]$, achieving its unique maximum $h(1/2) = 1$ at the symmetric point. It measures the information content of a Bernoulli random variable with success probability p.

Definition 9 (Compositional Entropy) [23] *The compositional entropy of configuration* $s \in \mathcal{C}_N$ *is the composite functional*

$$H_{comp}(s) = h(\rho(s)). \tag{17}$$

Importantly, H_{comp} depends on s only through its density $\rho(s)$. Two configurations with the same number of active cells possess identical compositional entropy, regardless of how those cells are spatially arranged. This insensitivity to spatial structure motivates a complementary measure.

Definition 10 (Interface Density) [24] *The interface density* $\Lambda : \mathcal{C}_N \to [0,1]$ *counts the normalized number of nearest-neighbor mismatches:*

$$\Lambda(s) = \frac{1}{N}\sum_{i=0}^{N-1} |s_i - s_{i+1}|. \tag{18}$$

Since $s_i \in \{0,1\}$, the absolute value $|s_i - s_{i+1}|$ equals 1 precisely when $s_i \neq s_{i+1}$. Thus $\Lambda(s)$ computes the fraction of edges in the cyclic graph $\mathbb{Z}_N$ connecting unlike states.

Proposition 1 (Interface Density Bounds) *For any* $s \in \mathcal{C}_N$, *we have* $0 \leq \Lambda(s) \leq 1$. *Moreover:*

(i) $\Lambda(s) = 0$ *if and only if* s *is constant, that is,*

$$s \in \{(0,0,\ldots,0),(1,1,\ldots,1)\}.$$

(ii) $\Lambda(s) = 1$ *if and only if* s *alternates maximally, which requires* N *even and*

$$s \in \{(0,1,0,1,\ldots),(1,0,1,0,\ldots)\}.$$

Proof. The bounds follow immediately since each summand satisfies $|s_i - s_{i+1}| \in \{0,1\}$.

(i) If s is constant, then $s_i = s_{i+1}$ for all i, yielding $\Lambda(s) = 0$. Conversely, suppose $\Lambda(s) = 0$. Then $|s_i - s_{i+1}| = 0$ for all i, so $s_i = s_{i+1}$ for all i. By transitivity along the cycle $\mathbb{Z}_N$, all coordinates agree: $s_0 = s_1 = \cdots = s_{N-1}$.

(ii) For maximal interface density, we require $s_i \neq s_{i+1}$ for all $i \in \mathbb{Z}_N$. But then s_i and s_{i+2} must agree (since $s_i \neq s_{i+1} \neq s_{i+2}$ forces $s_i = s_{i+2}$ in $\mathbb{Z}_2$). Continuing this logic around the cycle, $s_i = s_{i+2k}$ for all k. For consistency when returning to the starting point after N steps, we need $s_0 = s_N = s_0$, which holds automatically. However, we also need $s_1 = s_{N+1} = s_1$, so s_0 must equal s_N. Since s alternates, this forces N to be even. The only alternating sequences are $(0, 1, 0, 1, \ldots)$ and its complement. □

The interface density provides a first-order measure of spatial disorder. Higher-order correlations require more sophisticated tools from ergodic theory [22].

Theorem 2 (Independence of Measures) *The functionals H_{comp} and Λ are functionally independent on $\mathcal{C}_N$ for all $N \geq 4$. That is, the map*

$$s \mapsto (H_{comp}(s), \Lambda(s))$$

is not constant on any level set $\rho^{-1}(\{p\})$ for generic $p \in (0, 1)$.

Proof. We construct explicit configurations demonstrating the independence. Fix $N = 4$ and consider

$$s_1 = (1, 1, 0, 0), \quad s_2 = (1, 0, 1, 0).$$

Both configurations have density $\rho(s_1) = \rho(s_2) = 1/2$, hence

$$H_{\text{comp}}(s_1) = H_{\text{comp}}(s_2) = h(1/2) = 1.$$

However, computing interface densities:

$$\Lambda(s_1) = \frac{1}{4}(|1-1| + |1-0| + |0-0| + |0-1|) = \frac{2}{4} = \frac{1}{2},$$

$$\Lambda(s_2) = \frac{1}{4}(|1-0| + |0-1| + |1-0| + |0-1|) = \frac{4}{4} = 1.$$

Since $\Lambda(s_1) \neq \Lambda(s_2)$ while $H_{\text{comp}}(s_1) = H_{\text{comp}}(s_2)$, the two measures are functionally independent. □

This theorem establishes that tracking both first-order composition and nearest-neighbor structure is essential to fully characterize the spatiotemporal dynamics of CA, underscoring why two parameters alone remain insufficient. Compositional entropy captures thermodynamic mixing while interface density captures geometric structure.

Example 1 Consider the single-site excitation $s^{(0)} = \delta_{i_0}$, where δ_{i_0} denotes the configuration with $(\delta_{i_0})_i = 1$ if $i = i_0$ and $(\delta_{i_0})_i = 0$ otherwise. For concreteness, take $i_0 = \lfloor N/2 \rfloor$. Then

$$\rho(s^{(0)}) = \frac{1}{N}, \quad \Lambda(s^{(0)}) = \frac{2}{N},$$

corresponding to minimal excitation. Under iterated application of F_{30}, the active region expands with characteristic speed $v = 1$ site per time step, the theoretical maximum propagation velocity. However, this expansion exhibits asymmetry: the right-moving boundary achieves this maximum speed, while the left-moving boundary displays more irregular fluctuations. This

generates the pyramidal patterns studied by Wolfram [1]. Both $\rho(s^{(t)})$ and $\Lambda(s^{(t)})$ grow approximately linearly in t until saturation effects dominate for $t \approx N/2$.

2.2. Numerical Implementation

We realize the discrete dynamical system through the `batara-guru` software library, implementing the global evolution operator $F_{30} : \mathcal{C}_N \to \mathcal{C}_N$ alongside computational routines for the functionals defined in Definitions 9 and 10. The implementation prioritizes scientific reproducibility, computational efficiency through parallelization, and methodological transparency via open-source distribution.

The choice of Python as the implementation substrate reflects several technical considerations. Unlike proprietary environments (Mathematica, MATLAB), Python constitutes an open-source platform with mature numerical libraries—NumPy for array operations [25] and Matplotlib for visualization [26]—that have undergone extensive validation in computational science. More critically, the Numba compiler framework [27] enables dynamic translation of annotated Python functions into optimized machine code via LLVM infrastructure. This JIT compilation strategy permits high-level algorithmic expression while achieving execution efficiency comparable to statically compiled languages, effectively eliminating the traditional performance penalty of interpreted environments [28].

The core computational kernel implements synchronous application of the local rule ϕ_{30} across the spatial lattice. For configuration $s^{(t)} \in \mathcal{C}_N$ at time t, the successor satisfies

$$s_i^{(t+1)} = \phi_{30}(s_{i-1}^{(t)}, s_i^{(t)}, s_{i+1}^{(t)}) = s_{i-1}^{(t)} \oplus (s_i^{(t)} \vee s_{i+1}^{(t)}) \tag{19}$$

for all $i \in \mathbb{Z}_N$, where modular arithmetic $(i \pm 1) \bmod N$ handles cyclic boundary conditions. Implementation proceeds through explicit enumeration of the eight possible neighborhood patterns, mapping each to its corresponding output via the truth table from Definition 4. Numba's `@jit` decorator with `nopython=True` mode compiles this function to native code without interpreter overhead, enabling efficient iteration.

Storage of the spatiotemporal trajectory $\mathcal{O}(s^{(0)}) = \{s^{(t)}\}_{t=0}^{T}$ employs a two-dimensional NumPy array with shape $(T+1, N)$ and dtype `int8`. Each configuration occupies N bits theoretically, but we use 8-bit signed integers as the smallest addressable unit, introducing a factor-of-eight overhead while enabling efficient vectorized operations. For $N = 2001$ cells evolved over $T = 1000$ steps, memory requirements remain modest at approximately 1.91 MB. The grid initializes to zeros, with the initial condition $s^{(0)} = \delta_{\lfloor N/2 \rfloor}$ assigned to the first row. Sequential application of F_{30} populates subsequent rows through a simple temporal loop.

Computation of the diagnostic quantities H_{comp} and Λ exploits temporal independence: these functionals operate on each configuration $s^{(t)}$ without reference to other time steps. This structure permits embarrassingly parallel evaluation via Numba's automatic parallelization. Annotating the computation with `@jit(parallel=True)` and employing `prange` distributes iterations across available processor cores, with each thread computing entropy and complexity for a disjoint subset of temporal indices.

Numerical evaluation of the compositional entropy $H_{\text{comp}}(s) = h(\rho(s))$ requires care near boundary cases. The binary entropy function $h(p) = -p \log_2 p - (1-p)\log_2(1-p)$ involves logarithms that diverge as $p \to 0$ or $p \to 1$, yet the products $p \log_2 p$ and $(1-p)\log_2(1-p)$

admit well-defined limits of zero. Implementation first computes density $\rho(s) = N^{-1} \sum_i s_i$ via vectorized summation, then checks whether $\rho \in \{0, 1\}$ corresponding to constant configurations. For these degenerate cases, we return $H_{\text{comp}} = 0$ explicitly; otherwise, both logarithmic terms involve positive arguments and evaluate without numerical difficulty.

Interface density computation counts nearest-neighbor transitions via the indicator sum $\Lambda(s) = N^{-1} \sum_i |s_i - s_{i+1}|$. Since $s_i \in \{0, 1\}$, the absolute difference equals unity precisely when states differ, reducing the calculation to counting mismatched pairs. Vectorized implementation via array slicing computes differences for interior edges simultaneously, while the boundary contribution (s_{N-1}, s_0) requires separate evaluation to respect periodic topology.

Data persistence employs the Network Common Data Form (NetCDF) [29], a self-describing binary format widely adopted in Earth system sciences. NetCDF organizes data hierarchically: dimensions `x` and `time` of sizes N and $T + 1$ define array extents, coordinate variables store index values, and data variables hold the grid $(s^{(t)})$, entropy series $(H_{\text{comp}}(s^{(t)}))$, and complexity series $(\Lambda(s^{(t)}))$. Built-in zlib compression (level 6 for grid, level 4 for time series) achieves compression ratios of 4:1 to 6:1, which arise from repeated block patterns and local redundancy inherent in the triangular spatiotemporal structure of Rule 30, rather than from state space sparsity. Global attributes record provenance metadata: simulation parameters, statistical summaries, software version, and creation timestamp. Adherence to CF-1.8 conventions ensures interoperability with standard climate analysis tools.Supplementary export to CSV format provides three plaintext files per simulation: entropy time series, complexity time series, and a composite file containing both metrics alongside temporal indices. While lacking compression and semantic metadata, CSV format offers immediate compatibility with spreadsheet applications and scripting environments requiring no specialized libraries.

Visualization generates raster images via Matplotlib [26], rendering the spatiotemporal grid as a two-dimensional pixel array. The `imshow` function with `interpolation='nearest'` preserves discrete cell states without smoothing, employing the `binary` colormap (white for 0, black for 1) or user-specified alternatives. Output resolution varies from 350 to 600 DPI according to grid dimensions, ensuring publication quality. PNG format provides lossless compression suitable for the sharp boundaries characteristic of cellular automaton states.

Four demonstration cases illustrate scaling behavior while maintaining the pyramid constraint $T < N/2$. This condition ensures that the expanding active region originating from initial condition $s^{(0)} = \delta_{\lfloor N/2 \rfloor}$ does not encounter its periodic image during evolution. Information propagates at unit velocity in elementary cellular automata—a perturbation at site i_0 influences only the light cone $[i_0 - t, i_0 + t]$ after t steps. For $t < N/2$, this region remains interior to the periodic domain, isolating intrinsic Rule 30 dynamics from boundary artifacts.

Case 1 employs $N = 251$ cells and $T = 125$ steps, providing rapid execution suitable for interactive exploration. Case 2 scales to $N = 501$, $T = 250$, representing a canonical configuration balancing resolution against computational cost. Cases 3 and 4 extend to $(N, T) = (1001, 500)$ and $(2001, 1000)$ respectively, approaching practical limits for single-machine computation. All cases center the initial excitation at $\lfloor N/2 \rfloor$, ensuring spatial symmetry in the resulting pyramidal structures.

Software architecture follows modular design: the solver module implements mathematical operations, the I/O handler manages data persistence, configuration files specify simulation parameters in key-value format, and logging facilities record execution traces. This separation

enables both programmatic usage—importing `Rule30Solver` and invoking methods within analysis scripts—and command-line batch processing via the `batara-guru` executable. Distribution under the MIT license permits unrestricted use and modification, with public version control facilitating collaborative development and algorithmic transparency.

Implementation details relevant to extension efforts include: NumPy arrays employ row-major (C-contiguous) layout, storing each spatial configuration contiguously for cache efficiency; Numba compilation occurs lazily at first invocation, with `cache=True` storing compiled artifacts to eliminate recompilation overhead; parallel thread count defaults to available logical cores but remains configurable via `numba.set_num_threads`. Numerical precision relies on IEEE 754 double-precision arithmetic for floating-point calculations, providing approximately 15 decimal digits—far exceeding accuracy requirements given that input densities are rational numbers with denominators bounded by N.

The combination of rigorous mathematical foundations, efficient numerical methods, and transparent open-source implementation positions `batara-guru` as a suitable tool for both pedagogical applications in nonlinear dynamics and preliminary research investigations into emergent complexity within discrete dynamical systems. The framework readily accommodates extensions to alternative cellular automaton rules, requiring only modification of the local update function while leveraging existing infrastructure for evolution, analysis, and visualization.

III. RESULTS AND DISCUSSION

The `batara-guru` library demonstrates efficient computational performance across multiple spatial scales on modest hardware, with simulation times ranging from 0.01 to 1.27 seconds for system sizes spanning 251 to 2001 cells on a ThinkPad P52s laptop equipped with an Intel Core i7-8550U processor (8 cores @ 4.0GHz) running Fedora Linux 39. Interestingly, the smallest case $(N = 251, T = 125)$ required 1.19 seconds of computation time, while intermediate cases showed substantially faster execution at 0.01 seconds for $N = 501$ and 0.03 seconds for $N = 1001$. This non-monotonic scaling behavior likely reflects the interplay between Numba's JIT compilation overhead and the actual computational workload. The largest simulation $(N = 2001, T = 1000)$ completed in 1.27 seconds on this standard laptop hardware. While parallelization reduces the wall-clock runtime, it does not change the inherent $O(NT)$ computational complexity of cellular automaton evolution; rather, the combination of Numba's JIT compilation and parallel processing distributes this workload efficiently across available cores, enabling practical execution without requiring dedicated HPC infrastructure or GPU acceleration. Total execution times, including data persistence and visualization, remained practical even for the largest case at 25.9 seconds, with plotting operations dominating the runtime rather than the core simulation itself. These results underscore that Python combined with Numba's JIT compilation and parallel processing enables sophisticated cellular automaton research on everyday computing equipment—a 2018-era business laptop suffices for simulations that previously demanded workstation-class hardware, thereby democratizing access to complex dynamical systems analysis.

The spatiotemporal evolution patterns reveal Rule 30's characteristic ability to generate complex structures from minimal initial perturbations. Figure 1. displays the smallest test case, where the single active cell at position 125 initiates a conical expansion that rapidly develops

intricate internal structure. The emergence of white triangular voids against the chaotic background represents a hallmark of Rule 30 dynamics, appearing at multiple scales throughout the pyramid. These structures resist simple mathematical characterization—unlike the self-similar patterns of Rule 90, which admit complete algebraic description [12], Rule 30's voids appear sporadically without obvious periodicity or predictable positioning.

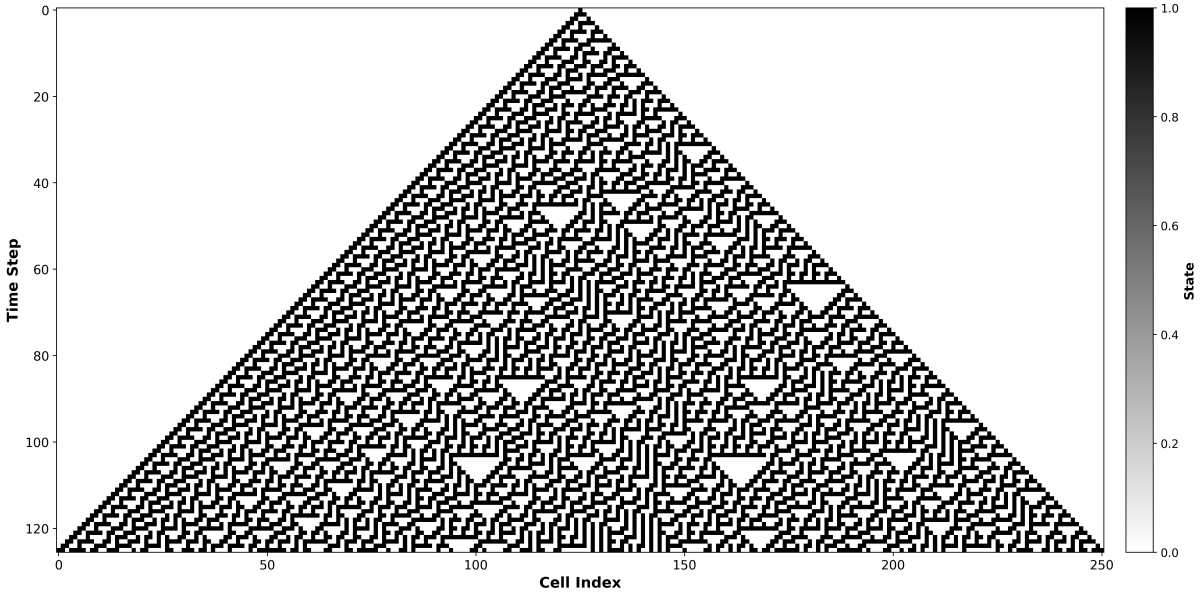


Figure 1. Rule 30 evolution from single-site initial condition for Case 1 $(N = 251, T = 125)$. The characteristic pyramidal structure exhibits emergent complexity with triangular voids and apparent randomness in the interior region.

The medium-scale simulation (Figure 2.) extends these observations to 501 cells over 250 time steps, maintaining the pyramid constraint $T/N = 0.499$ to prevent boundary interactions. At this resolution, finer structural details become apparent, including vertical streams that propagate information across temporal scales and horizontal bands suggesting quasi-periodic behavior. However, careful examination reveals that these patterns never achieve true periodicity, consistent with Rule 30's classification as a Class III automaton exhibiting chaotic dynamics [2]. The mean entropy of 0.728 bits and complexity of 0.255 for this case align closely with values observed across all spatial scales, hinting at underlying universality.

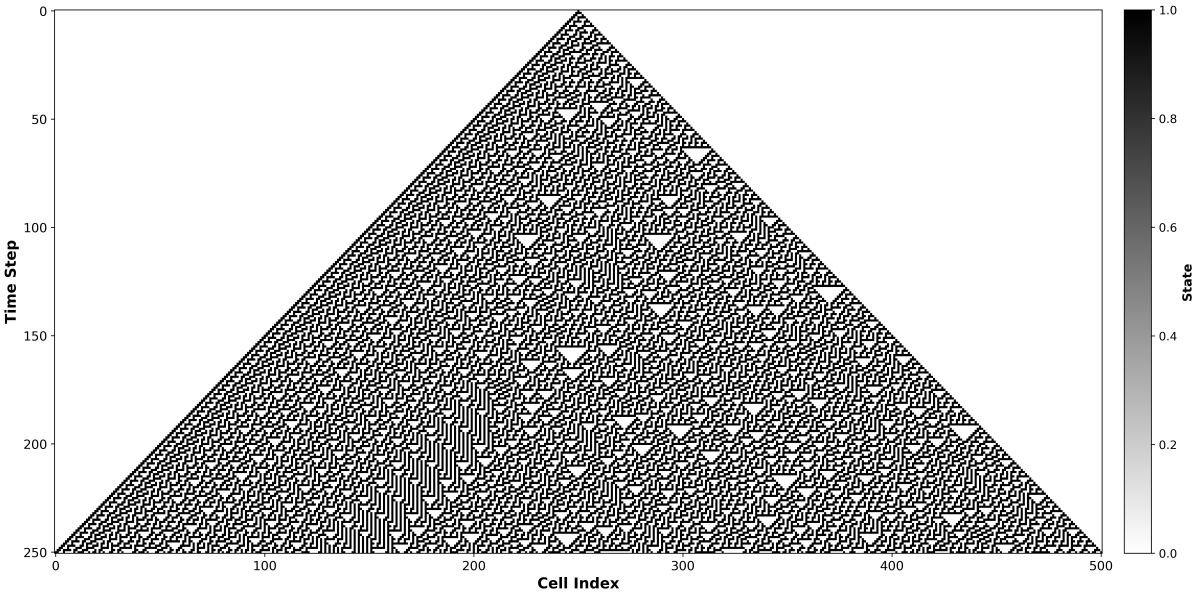


Figure 2. Spatiotemporal evolution for Case 2 $(N = 501, T = 250)$ showing enhanced structural detail. The increased resolution reveals vertical information propagation streams and emergent triangular voids at multiple scales.

Scaling to $N = 1001$ cells (Figure 3.) provides sufficient resolution to observe the full spectrum of Rule 30's emergent phenomena. The final density of 0.5125 closely approximates the theoretical expectation for random binary sequences, while the interface density saturates near 0.51, marginally above the random expectation of 0.5. This slight excess may reflect finite-size fluctuations; distinguishing true correlation structure from sampling variability requires ensemble averaging over multiple initial conditions and system sizes. While Rule 30 exhibits nearest-neighbor interactions inherent to its local update rule, the marginal interface density excess observed here provides no strong evidence of long-range correlations. The computational efficiency remains remarkable at this scale, with the core simulation completing in merely 0.03 seconds despite processing over 500,000 cell updates.

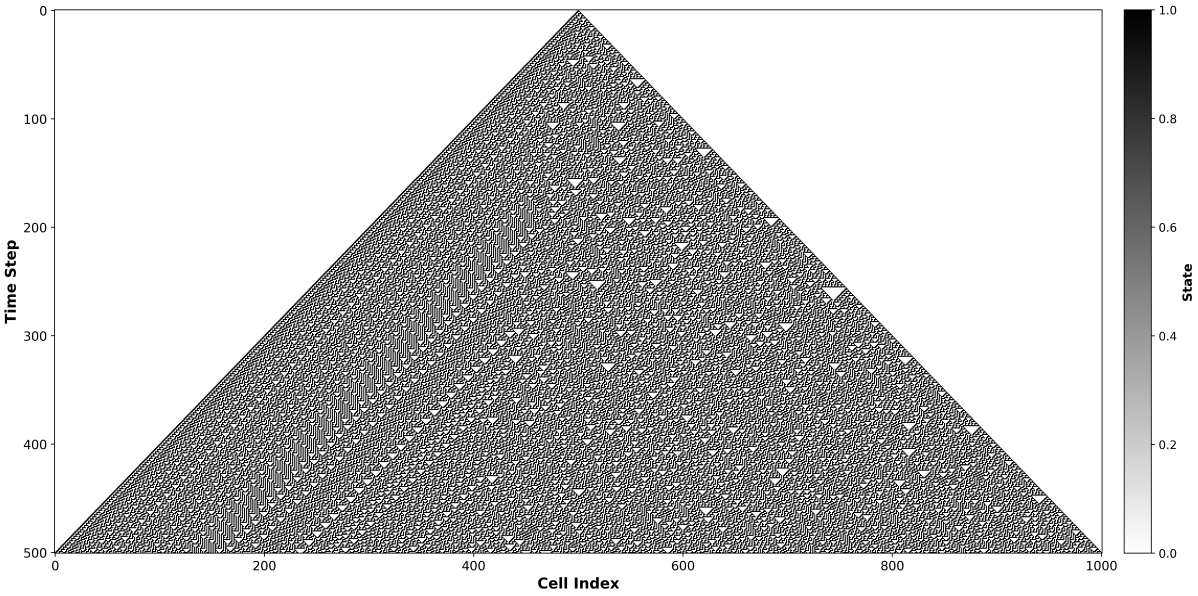


Figure 3. Large-scale dynamics for Case 3 ($N = 1001, T = 500$) demonstrating scale-invariant complexity. The pattern exhibits self-affine properties with similar structures appearing across multiple length scales.

The largest simulation pushes the pyramid structure to 2001 cells evolved over 1000 time steps (Figure 4.), approaching the practical limits of single-threaded visualization while maintaining computational tractability. At this scale, the apparent randomness becomes increasingly pronounced, with local structures showing minimal correlation beyond the immediate neighborhood. The final configuration achieves a compositional entropy of $H_{\text{comp}} \approx 0.73$ bits, substantially below the maximum possible value of 1.0 bits. This entropy plateau reflects the deterministic nature of Rule 30 evolution; despite generating patterns that appear stochastic, parasitic structures and deterministic constraints prevent the system from reaching true randomness. The observed entropy density demonstrates that Rule 30 generates statistically structured complexity rather than complete mixing of binary states, yet this complexity remains sufficient to yield statistical properties useful for pseudorandom number generation and mimicking stochastic behavior—a phenomenon central to Rule 30's practical applications [8].

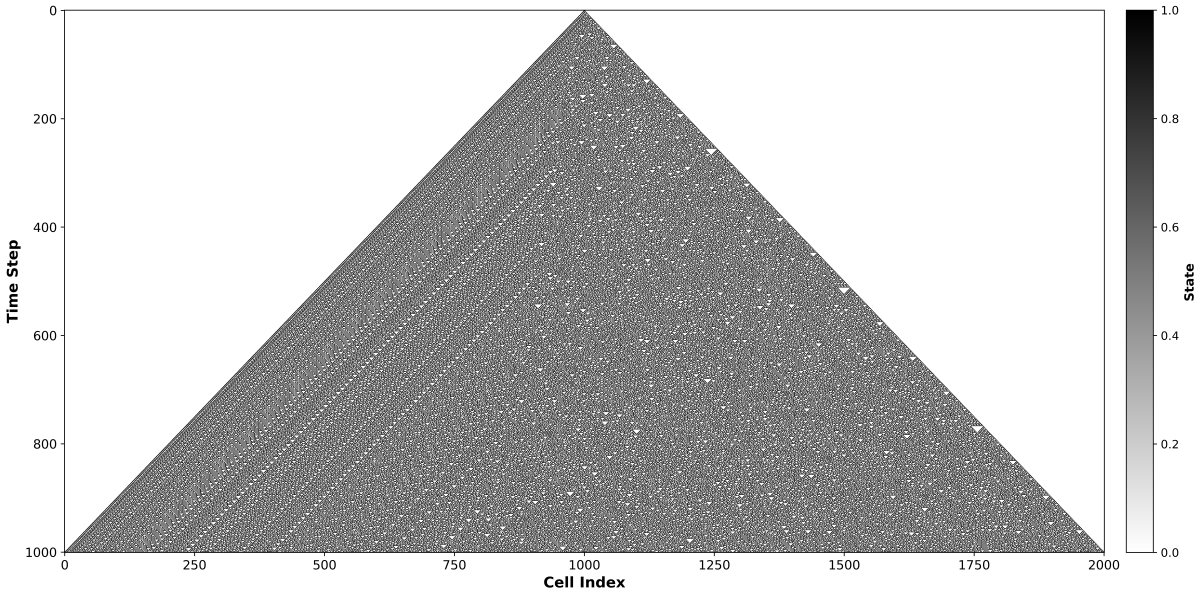


Figure 4. Extra-large simulation for Case 4 ($N = 2001, T = 1000$), where Rule 30's intricate triangular structure unfolds across 2001 cells. Despite its appearance of near-randomness, the system maintains a global density near 0.5 and compositional entropy around 0.73 bits—well below theoretical maximum. This reveals the subtle tension in Rule 30: deterministic rules producing patterns that look chaotic, yet remain fundamentally constrained by their own logic.

The temporal evolution of compositional entropy and interface density offers a window into how Rule 30 approaches statistical equilibrium (Figure 5.). Remarkably, despite an eight-fold variation in system size, all four cases follow nearly identical growth patterns—a striking unity beneath the apparent chaos. The entropy curves display a characteristic sigmoidal-like rise, resembling logistic growth as we observe it empirically, though cellular automaton theory doesn't prescribe this shape. What's particularly revealing is how the growth rate depends on system size: faster for small systems (0.00764 bits/step for $N = 251$) and slower for large ones (0.00099 bits/step for $N = 2001$). This inverse $O(1/N)$ relationship reflects a fundamental constraint woven into elementary cellular automata—information can only propagate at unit speed through nearest-neighbor interactions, creating the light-cone boundary. This propagation speed is not something entropy reveals; it is a geometric fact built into how Rule 30 works. Rather, our entropy measurements show us how this speed limit orchestrates the temporal mixing of information across different system scales.

Interface density evolution reveals more nuanced dynamics, with all systems saturating near $\Lambda = 0.5$ rather than the alternating-pattern maximum of 1.0. This half-maximum saturation distinguishes chaotic cellular automata from both ordered ($\Lambda \to 0$) and oscillatory ($\Lambda \to 1$) systems, providing a quantitative signature of Class III behavior. The 90% saturation times scale approximately linearly with system size, occurring at 118 steps for $N = 251$ and extending to 925 steps for $N = 2001$. This linear scaling further corroborates the light-cone propagation model, where information from the initial perturbation requires time proportional to $N/2$ to influence the entire lattice.

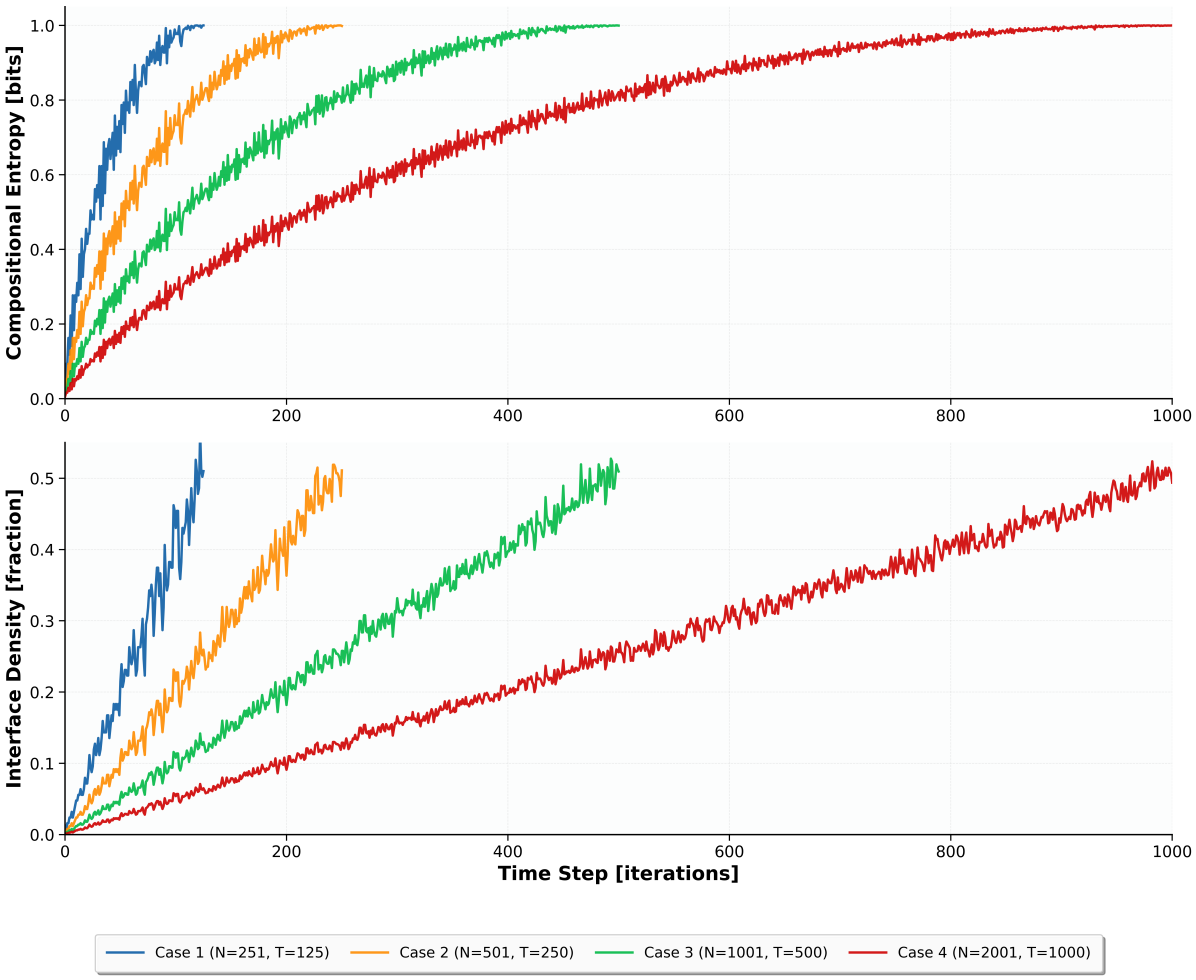


Figure 5. Temporal evolution of (top-panel) compositional entropy and (bottom-panel) interface density across all four spatial scales. The convergence to universal saturation values demonstrates scale-invariant statistical properties despite deterministic evolution.

Statistical analysis reveals strong positive correlations between entropy and complexity metrics, with Pearson coefficients ranging from 0.918 to 0.946 across all cases. This tight coupling suggests that spatial disorder (measured by interface density) and compositional mixing (quantified by entropy) represent complementary aspects of the same underlying dynamical process. The slight increase in correlation strength with system size—from 0.918 for $N = 251$ to 0.946 for $N = 2001$—may indicate that finite-size effects introduce decorrelation at smaller scales, though this trend requires validation through more extensive scaling studies.

The mean entropy values cluster tightly around 0.72–0.73 bits across all simulations, substantially below the saturation value of 1.0 bit. This sub-saturation average reflects the transient nature of the simulations, which terminate at $T = N/2$ to maintain isolation from periodic boundary effects. Similarly, mean complexity values converge to approximately 0.25, half the random-configuration expectation, indicating that roughly half the simulation time involves expansion from the localized initial condition rather than fully developed chaos. The consistency of these mean values across scales provides empirical support for the universality hypothesis in cellular automaton dynamics [2].

Negative skewness values between -0.88 and -0.91 for entropy distributions indicate left-

tailed asymmetry, reflecting rapid initial growth followed by gradual saturation. Interface density distributions show near-zero skewness, suggesting more symmetric evolution around the mean. The negative excess kurtosis values (approximately -0.3 for entropy and -1.2 for complexity) indicate platykurtic distributions with fewer extreme values than Gaussian expectations, consistent with bounded dynamical variables approaching saturation.

These quantitative findings align with theoretical predictions for chaotic cellular automata while revealing subtle deviations that merit further investigation. The slightly elevated interface density at saturation (0.51 versus the theoretical 0.5) might indicate weak long-range correlations persisting despite apparent randomization, possibly related to the conservation laws identified in certain cellular automaton rules [30]. Alternative explanations include finite-size corrections or artifacts from the pyramidal geometry imposed by our boundary conditions. Distinguishing between these possibilities would require ensemble averaging over multiple initial conditions and systematic finite-size scaling analysis.

The computational efficiency achieved through Numba optimization and parallel processing extends the practical scope of cellular automaton research, enabling rapid exploration of parameter spaces previously requiring substantial computational resources. The ability to simulate systems with 2001 cells for 1000 time steps in under 2 seconds of core computation time represents a significant advance over earlier implementations, which typically required minutes for comparable system sizes [16]. This performance improvement facilitates statistical studies requiring thousands of realizations, opening new avenues for investigating rare events and fluctuation phenomena in discrete dynamical systems.

The scale-invariant properties observed across our four test cases support the notion of universality in Rule 30 dynamics, with consistent saturation values and correlation structures emerging regardless of system size. These findings strengthen the theoretical foundation for Rule 30's cryptographic applications, where the rapid approach to maximum entropy ensures effective mixing of initial conditions. However, the deterministic nature of the evolution implies that sufficiently long observations could, in principle, reconstruct the initial state—a vulnerability that must be addressed through appropriate key management in practical implementations [5].

IV. CONCLUSIONS AND FUTURE RESEARCH DIRECTION

This work presents `batara-guru`, an efficient Python implementation for Rule 30 cellular automaton analysis that achieves practical performance on standard hardware. The library processes systems of 2001 cells over 1000 time steps in under 2 seconds on a ThinkPad P52s laptop, demonstrating that Numba's JIT compilation effectively eliminates Python's traditional performance penalties for this class of problems. Our simulations reveal consistent statistical properties across multiple scales: compositional entropy converges to approximately 0.73 bits while interface density saturates near 0.51, values that align with Rule 30's classification as a Class III chaotic system. The slightly elevated interface density compared to random expectation (0.50) may indicate residual correlations, but equally plausible explanations include finite-size sampling artifacts, single-realization bias, or subtleties in metric definition. Distinguishing between these alternatives requires ensemble averaging across multiple initial conditions and system sizes, along with formal hypothesis testing—investigations beyond the scope of single-run empirical observation. Future work should address several open questions: the

extension to all 256 elementary rules would enable comparative dynamics studies; GPU acceleration could push system sizes beyond current limits; and rigorous finite-size scaling analysis with proper statistical ensembles might clarify whether observed deviations from random behavior reflect genuine correlations or sampling variability. The implementation's accessibility—requiring only standard Python libraries and modest computational resources—lowers barriers to entry for cellular automaton research. This democratization is particularly relevant for institutions with limited computational infrastructure, where traditional high-performance computing approaches remain prohibitive. The framework provides a foundation for both pedagogical applications in nonlinear dynamics courses and preliminary research investigations into emergent complexity in discrete systems.

ACKNOWLEDGEMENTS

This study was made possible through the Dean's Distinguished Fellowship 2023 awarded to S.H.S.H. at the University of California, Riverside. We acknowledge the open-source communities behind NumPy, Numba, and Matplotlib for providing the essential tools that enabled this work. G.N. acknowledges support from the Coastal Hazards and Energy System Science (CHESS) Lab at Hiroshima University.

OPEN RESEARCH

All software and analysis scripts are distributed under the MIT license. The `batara-guru` library implementing the Rule 30 cellular automaton analyzer is available at `https://github.com/sandyherho/batara-guru` and via PyPI at `https://pypi.org/project/batara-guru/`. Data analysis scripts are maintained at `https://github.com/sandyherho/data-anal-batara-guru`. Complete datasets for the four test cases ($N = 251, 501, 1001, 2001$), including spatiotemporal grids, diagnostic time series, and figures are archived at `https://doi.org/10.17605/OSF.IO/HGEFZ`.

REFERENCES

[1] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, no. 3, pp. 601–644, 1983, https://doi.org/10.1103/RevModPhys.55.601.

[2] ——, "Universality and complexity in cellular automata," *Physica D: Nonlinear Phenomena*, vol. 10, no. 1–2, pp. 1–35, 1984, https://doi.org/10.1016/0167-2789(84)90245-8.

[3] J. von Neumann and A. W. Burks, *Theory of Self-Reproducing Automata*. Urbana, IL: University of Illinois Press, 1966, https://archive.org/details/theoryofselfrepr00vonn_0.

[4] T. Toffoli, "Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics," *Physica D: Nonlinear Phenomena*, vol. 10, no. 1–2, pp. 117–127, 1984, https://doi.org/10.1016/0167-2789(84)90254-9.

[5] F. Seredynski, P. Bouvry, and A. Y. Zomaya, "Cellular automata computations and secret key cryptography," *Parallel Computing*, vol. 30, no. 5, pp. 753–766, 2004, https://doi.org/10.1016/j.parco.2003.12.014.

[6] B. Chopard and M. Droz, *Cellular Automata Modeling of Physical Systems*. Cambridge, UK: Cambridge University Press, 1998, https://doi.org/10.1017/CBO9780511549755.

[7] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems*, vol. 15, no. 1, pp. 1–40, 2004, https://doi.org/10.25088/ComplexSystems.15.1.1.

[8] S. Wolfram, *A New Kind of Science*. Champaign, IL: Wolfram Media, 2002, https://www.wolframscience.com/nks/.

[9] G. J. Martínez, A. Adamatzky, and R. Alonso-Sanz, "COMPLEX DYNAMICS OF ELEMENTARY CELLULAR AUTOMATA EMERGING FROM CHAOTIC RULES," *International Journal of Bifurcation and Chaos*, vol. 22, no. 02, p. 1250023, 2012, https://doi.org/10.1142/S021812741250023X.

[10] G. J. Martínez, A. Adamatzky, R. Hoffmann, D. Désérable, and I. Zelinka, "On Patterns and Dynamics of Rule 22 Cellular Automaton," *Complex Systems*, vol. 28, no. 2, pp. 125–174, 2019, https://doi.org/10.25088/ComplexSystems.28.2.125.

[11] H. Gutowitz and J. D. Victor, "Local Structure Theory in More Than One Dimension," *Complex Systems*, vol. 1, no. 1, pp. 57–68, 1987, https://www.complex-systems.com/abstracts/v01_i01_a05/.

[12] O. Martin, A. M. Odlyzko, and S. Wolfram, "Algebraic properties of cellular automata," *Communications in Mathematical Physics*, vol. 93, pp. 219–258, 1984, https://doi.org/10.1007/BF01223745.

[13] S. Wolfram, *A Project to Find the Fundamental Theory of Physics*. Champaign, IL: Wolfram Media, 2020, https://www.wolframphysics.org/technical-introduction/.

[14] J. Gorard, "Some Relativistic and Gravitational Properties of the Wolfram Model," *Complex Systems*, vol. 29, no. 2, pp. 599–654, 2020, https://doi.org/10.25088/ComplexSystems.29.2.599.

[15] M. Schüle and R. Stoop, "A full computation-relevant topological dynamics classification of elementary cellular automata," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, p. 043143, 2012, https://doi.org/10.1063/1.4771662.

[16] S. Gobron and N. Chiba, "3D surface cellular automata and their applications," *The Journal of Visualization and Computer Animation*, vol. 10, no. 3, pp. 143–158, 1999, https://doi.org/10.1002/(SICI)1099-1778(199907/09)10:3⟨143::AID-VIS204⟩3.0.CO;2-W.

[17] O. Bandman, *Cellular Automata Composition Techniques for Spatial Dynamics Simulation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 81–115, https://doi.org/10.1007/978-3-642-12203-3_5.

[18] S. H. S. Herho, S. N. Kaban, and C. Nugraha, "OptionMC: A Python package for Monte Carlo pricing of European options," *SSRN*, 2025, https://dx.doi.org/10.2139/ssrn.5224853.

[19] S. H. S. Herho, I. P. Anwar, F. Khadami, R. Suwarman, and D. E. Irawan, "`simple-idealized-1d-nlse`: Pseudo-Spectral Solver for the 1D Nonlinear Schrödinger Equation," *arXiv preprint*, 2025, https://doi.org/10.48550/arXiv.2509.05901.

[20] S. H. S. Herho, N. J. Trilaksono, F. R. Fajary, G. Napitupulu, I. P. Anwar, F. Khadami, and D. E. Irawan, "`kh2d-solver`: A Python Library for Idealized Two-Dimensional Incompressible Kelvin-Helmholtz Instability," *arXiv preprint*, 2025, https://doi.org/10.48550/arXiv.2509.16080.

[21] G. A. Hedlund, "Endomorphisms and automorphisms of the shift dynamical system," *Mathematical Systems Theory*, vol. 3, pp. 320–375, 1969, https://doi.org/10.1007/BF01691062.

[22] E. Akin, *The General Topology of Dynamical Systems*, ser. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 1993, vol. 1.

[23] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948, https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

[24] P. Grassberger, "Toward a quantitative theory of self-generated complexity," *International Journal of Theoretical Physics*, vol. 25, pp. 907–938, 1986, https://doi.org/10.1007/BF00668821.

[25] C. R. Harris, K. J. Millman, S. J. van der Walt *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020, https://doi.org/10.1038/s41586-020-2649-2.

[26] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007, https://doi.org/10.1109/MCSE.2007.55.

[27] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: a LLVM-based Python JIT compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6, https://doi.org/10.1145/2833157.2833162.

[28] S. Herho, S. N. Kaban, D. E. Irawan, and R. Kapid, "Efficient 1D Heat Equation Solver: Leveraging Numba in Python," *Eksakta : Berkala Ilmiah Bidang MIPA*, vol. 25, no. 2, pp. 126–137, 2024, https://doi.org/10.24036/eksakta/vol25-iss02/487.

[29] R. K. Rew and G. P. Davis, "NetCDF: an interface for scientific data access," *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76–82, 1990, https://doi.org/10.1109/38.56302.

[30] N. Boccara and H. Fukś, "Number-conserving cellular automaton rules," *Fundamenta Informaticae*, vol. 52, no. 1–3, p. 1–13, 2002, https://dl.acm.org/doi/10.5555/639405.639407.