ChatGPT

Identifying Interesting Life-Like CA Rules: Metrics and Methods

Researchers have proposed many measures and techniques to screen CA rules for nontrivial behavior. Broadly, **statistical or information-theoretic metrics** are used to quantify "complexity" (e.g. entropy, Kolmogorov complexity), while **pattern-based metrics** look for symmetry or emergent structures (gliders, periodicity, etc.). Below we survey key approaches from the literature and discuss modern ML and clustering methods, along with practical considerations.

1. Metrics for Complex or "Interesting" Behavior

- **Shannon and related entropies:** One of the simplest metrics is the spatial/temporal Shannon entropy of the CA state. For example, Peña & Sayama (2021) track average density and **entropy** (or conditional entropy) over time to characterize Life-like rules [1]. Low entropy indicates periodic or static patterns; very high entropy indicates randomness. Intermediate entropy often correlates with complex structures. Similarly, **input-entropy variance** (Wuensche 1998) measures the variance of the local neighborhood entropy over time – this sharply separates Wolfram's classes in 1D CAs and can "screen out" trivially ordered or chaotic rules [2]. In practice one computes, say, the entropy of the neighborhood configuration distribution at each step and then the variance of that series; high variance typically signals glider-rich (Class 4) behavior [2].

- **Algorithmic/Compression complexity:** Another widely used idea is to approximate **Kolmogorov complexity** via compression. For a given rule and initial state, form the space-time diagram (e.g. a 2D image or flattened string) and measure its compressed length. Chaotic patterns compress poorly (large output), very regular patterns compress well (small output); interesting rules often lie in between. Hector Zenil et al. showed that clustering CA rules by Lempel–Ziv (gzip) compression of their output yields clusters matching Wolfram classes [3]. For example, in ECA the scatterplot of compressed lengths (Figure below) separates Class 1/2 rules (low length, green points) from Class 3 (high length, orange/blue) and Class 4 (intermediate, shown in green) [4]. (Images below from Cisneros *et al.*, 2019 [4].)

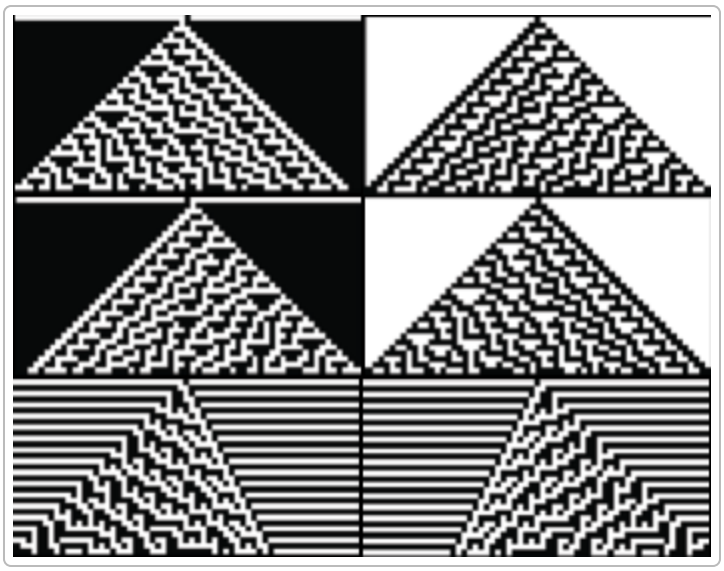

Figure: Space-time outputs of the "highest-complexity" 1D CA rules (e.g. Rule 110) and their neighbors [4] .

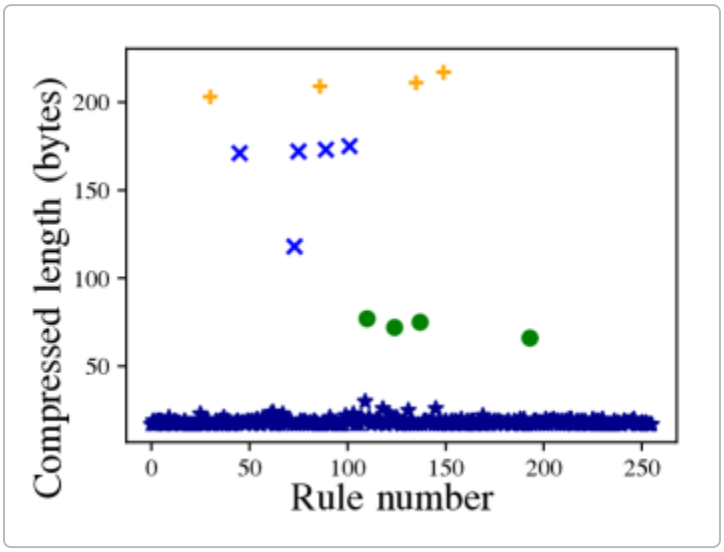

Figure: Compressed-length of all 256 elementary CA rules. Points cluster by behavioral class (blue = chaotic, green = complex, orange = ordered) [4] .

These "compression clusters" show that mid-range complexity rules occupy a distinctive band [4] . However, compression alone misses the *dynamics* – for example, a rule that quickly randomizes will compress nearly maximally but is not interesting in the Wolfram sense. Cisneros *et al.* therefore propose a **predictor/joint-compression metric** that measures how complexity grows over time (e.g. comparing the compressed length of successive frames) to emphasize sustained structure [5] .

- **Statistical pattern metrics:** Simpler statistics can also help flag interesting behavior. For instance, **cell density** or fraction of live cells is often tracked (Peña & Sayama use average density on [0,1]) [1] . The *variance* or *fluctuations* of density can indicate periodic patterns. One can also compute the **autocorrelation in time** or the **Fourier spectrum** of the total live-cell count to detect hidden periodicities or oscillations. Likewise, spatial measures like the number of connected clusters or

perimeter/area ratios can quantify “structure” vs. uniform noise. In effect, interesting Life-like rules tend to maintain nontrivial patterns (gliders, oscillators) and not simply fill the grid, so sudden convergence to all-dead or all-alive (trivial attractors) scores low on these measures.

- **Emergent structure counts:** Arguably the most direct indicators are counts of **gliders, oscillators, replicators**, etc. Wuensche’s work on Life (2D) illustrates that gliders appear as diagonal stripes in the spacetime stack [6]. Automated detectors (e.g. convolutional filters matched to known glider shapes or cross-correlation with pattern libraries) can thus identify emergent mobile structures. For example, the Softology blog describes trying filters or glider templates to pick out interesting dynamics [7]. More generally, one can measure the **pattern diversity** (e.g. number of distinct local neighborhoods that appear) or the **Lyapunov exponent** (sensitivity to initial perturbations). No single metric is definitive, but combinations (entropy + pattern count + periodicity) tend to be effective.

- **Symmetry and invariance:** Some metrics exploit rule symmetries. For instance, Life-like rules are isomorphic under **ON/OFF inversion** (“Day & Night” rule) and under alternating generation inversion (“strobing”) [8]. When scanning the rule space, one can identify symmetric equivalents and only test one representative. In terms of patterns, symmetry (rotational or reflectional) is usually aesthetic but not inherently “complex” by Wolfram’s criteria; thus there is no standard measure of symmetry used for classification. (One approach detects axes of symmetry via swarm algorithms [9], but this is more for evolved art than CA search.)

In practice, researchers often combine metrics. For example, Liao *et al.* (2021) used both **statistical features** (like entropy of spatiotemporal images) and CNN-extracted features to distinguish “boring” vs. “interesting” rules [10]. In summary, proven metrics include entropy (Shannon and block/conditional), input-entropy variance [2], compression/LZ complexity [3], and counts of emergent structures (gliders/oscillators) as diagnostics. These capture different aspects (randomness vs. order vs. organized structure).

2. Deep Learning Approaches (CNNs, RNNs, GNNs)

Recent work has applied modern neural networks to CA classification:

- **Convolutional Neural Networks (CNNs):** Treat the CA spacetime diagram as an image (time on one axis, space on the other) and train CNNs to recognize patterns or classify behavior. For example, Rollier *et al.* (2024) showed that a CNN can *supervisedly* classify elementary CA into Wolfram classes by learning the global texture of the spacetime diagram [11]. Similarly, Liao’s Berkeley project (2021) used CNNs on “stitched” CA frames to detect whether a rule produces spaceships/gliders [10]. Their CNN achieved high accuracy (~84% test accuracy with 100% recall on “interesting” cases) on a hand-labeled dataset [12]. In training, data augmentation (rotations, flips) helps the CNN focus on structure rather than absolute orientation. Evaluation is via standard metrics (accuracy, recall) on held-out rule samples [12].

- **Recurrent Neural Networks (RNNs/LSTMs):** Since CA produce sequences of configurations, RNNs can be applied to classify the *temporal evolution*. Liao *et al.* found that LSTM-based RNNs (on sequences of CA images) can also learn to flag complex rules [10]. Another approach (Aach, 2023) trained a CNN+encoder-decoder to *predict* the next state of 2D CA from previous states; it achieved

~93% accuracy for known rules and ~77% on unseen rules [13]. While that work focused on learning the update rule, it demonstrates that neural nets can capture CA dynamics and could be repurposed to classify behaviors.

- **3D CNNs and Spatiotemporal CNNs:** In principle, one can use 3D convolutions (treating time as a dimension like video) or ConvLSTMs to capture spatiotemporal features. Some hybrid CA models (in land-use simulation) use 3D-CNN+LSTM architectures [14]. In principle, one could stack several time-steps and feed into a 3D CNN to let it learn motion patterns (gliders, waves). However, most CA classification efforts so far use 2D CNNs on images of space-time (see above) rather than explicit 3D networks.

- **Graph Neural Networks (GNNs):** GNNs have been applied in **Graph Cellular Automata**, a generalization where cells form a graph. For regular grids, GNNs could model local interactions. Recent work ("Learning Graph Cellular Automata", NeurIPS 2021) uses GNNs to learn arbitrary CA update rules, but not specifically to classify rule behavior [15]. We are not aware of successful GNN-based classifiers targeting CA "interestingness". However, GNNs have potential if one represents the grid as a graph and learns dynamics; this is more often used to simulate CA rather than analyze them.

- **Neural CA and Differentiable Models:** Relatedly, neural-CA models (where a network *is* the update rule) have become popular. For example, Mordvintsev *et al.* (2020) introduced "Growing Neural Cellular Automata" which use CNNs to learn Life-like rules that *regenerate* target patterns [16]. Recently, Google's "Differentiable Logic CA" (2025) combined neural networks with discrete logic to learn rules that produce desired patterns [17]. While these works are generative (designing CA to achieve a pattern), their techniques (neural networks, autoencoder modules) are relevant. One could imagine using a Neural CA framework to learn the mapping from rule parameters to "interestingness" score and use gradient-based search.

In all cases, training is **supervised** (requiring labeled examples of interesting vs boring rules or labeled classes). Labels can come from domain experts (known Life-like rules, glider-rich examples) or heuristics. Results are then validated on held-out rules or by discovering new complex rules: notably, Liao *et al.* report that their trained networks indeed found novel spaceship-producing rules (including a "new Life" variant) [10].

3. Clustering by Simulation Behavior

Rather than outright classification, one can **cluster rules by their simulated behavior**. The idea is to embed each rule's dynamics into a feature space (so that "similar-looking" evolutions map close together) and apply unsupervised clustering. Key points:

- **State representation:** A rule's behavior can be summarized by one or more output configurations. For example, take the **final state** after N steps, or the entire sequence of states (a binary 3D tensor of size [T×W×H]). Flattening this tensor yields a high-dimensional binary vector. In practice, one can downsample (coarse-grain) or use summaries (density over time, spatial autocorrelation) to reduce dimensionality. Cisneros *et al.* (2020) propose coarse-graining by grouping cells based on their activity frequency and using autoencoders to compress large space-time outputs [18] [19].

- **Dimensionality reduction and autoencoders:** Autoencoders or embedding networks can learn compact representations of space-time diagrams. For example, the MIT/ALife 2020 paper "Visualizing Computation in Large-Scale CA" uses autoencoders to reduce huge 2D CA evolutions to smaller feature maps [19]. One could train an autoencoder on many CA outputs and then cluster in the latent space. Other approaches: principal component analysis (PCA) or t-SNE/UMAP on feature vectors (e.g. flatten + PCA of final states) can group similar behaviors.

- **Clustering algorithms:** Common methods (K-means, DBSCAN, hierarchical clustering) can be applied on the chosen features. As an example, Cisneros *et al.* clustered elementary CA by the compressed length metric [4] (as discussed above) to find groups of similar dynamics. One could similarly cluster by *multiple metrics* – e.g. use [density, entropy, Lyapunov exponent] as a 3D feature and apply K-means.

- **Trajectory distance metrics:** Another idea is to define a distance between two rules based on how their states evolve. For instance, one could run both rules on the same random seed and measure the Hamming distance between the two spacetime patterns. A low distance implies similar behavior. More sophisticated: use dynamic time warping or Edit-distance on time series of pattern statistics. (This is rarely done in existing CA work, but is an option.)

- **Example – compression-based clustering:** The figures above illustrate one clustering approach. Cisneros *et al.* plotted the compressed lengths of all 256 ECA rules [4]. Rules naturally fell into bands: Class 3 (chaotic) with high lengths (orange/blue in the figure) and Class 1/2 (stable) with low lengths (green/blue). A K-means on those lengths recovered Wolfram's classes [4]. This suggests that even a 1D feature (compressed length) can separate broad behavior types.

- **Representation tips:** In practice, for Life-like CAs one might represent each rule by the final density + entropy + count of live blobs after T steps, or by a small collection of snapshots (e.g. at t= T/2 and t=T). One can also encode each state image via pretrained CNN feature vectors. The choice depends on what distinguishes interesting rules: if it's "glider complexity", features sensitive to moving structures (e.g. local pattern frequency) should be used.

Is clustering promising? Yes, especially as a *pre-screening*. Clustering can highlight pockets of complex behavior that merit human inspection. However, unsupervised clusters still need interpretation: a cluster may just contain many trivial rules if the features are not well-chosen. Combining clustering with visual inspection (e.g. plotting representative patterns of each cluster) can help. In summary, clustering of simulation outputs is a valid approach, but its success hinges on using a rich representation of the CA behavior (see above) that captures spatial-temporal structure.

4. Balancing Efficiency and Long-Term Behavior

Simulating all 2^18 ≈ 262K Life-like rules is computationally heavy, especially if each is run for thousands of steps. Some considerations:

- **Grid size vs. time steps:** The user's pipeline uses a 30×30 grid for ~1000–2000 steps per rule. Larger grids (e.g. 50×50 or 100×100) may reveal more gliders or interactions, but cost scales as area × steps. If resources are limited, one can simulate smaller grids (as here) but accept missing some large-

structure behaviors. Adaptive schemes can help: e.g. run all rules for a short time (say 100 steps) to eliminate trivial cases (immediate extinction or fill), then continue *only* on the remaining ~10–20% that survived with moderate activity.

- **Multiple initial conditions:** A single random seed may not reveal a rule's full complexity. Some work (Peña & Sayama) uses many random runs per rule (e.g. 50 runs of length 2000) to estimate average behavior [1]. To save time, one could use fewer runs but a variety of seeds for each rule, or focus on seeds that avoid premature death (e.g. start with half density).

- **Early stopping and filters:** If a CA quickly enters a fixed point or all-dead state, stop the simulation for that rule early. Conversely, if the pattern already looks fully chaotic (entropy saturating at max), one might also stop. Thus implement checks each few steps: if the state stops changing or becomes all alive/dead, mark the rule as trivial and halt. This avoids wasting steps on obviously boring rules.

- **Approximate or multi-scale methods:** One could simulate coarsely (e.g. update blocks of cells at once) for preliminary scanning. Cisneros *et al.* note that coarse-graining (merging 2×2 blocks) can reveal large-scale emergent features [18]. GPU or bitwise-parallel CA libraries (like the CARLE simulator [20]) can also greatly speed up batch runs. Indeed, the CARLE environment is optimized to simulate thousands of Life-like rules in parallel [20]; using such tools (if available) can make exhaustive scanning feasible.

- **Long-term vs. transient:** Some interesting rules reveal gliders only after hundreds of steps. If only short simulations are used, one might miss these. A compromise is to sample a few long runs for rules that appear promising early on. For example, after 100–200 steps, if a rule shows moderate live density and not chaotic noise, then run it to 1000–2000 steps to see if gliders emerge. This two-phase approach (short filter, then extended run) balances speed and thoroughness.

In summary, it is important to tune simulation parameters. The Peña & Sayama study itself ran 50 trials of 2000 steps per rule [1], highlighting that long runs are valuable for stable statistics. If the current pipeline finds too few complexes, consider increasing run length or grid size (if compute permits) or adding multiple seeds, coupled with the above stopping heuristics.

5. Hybrid (Statistical + ML) Approaches

Combining traditional metrics with machine learning often yields the best results. For example:

- **Feature+Classifier:** Compute a set of statistical features (entropy, density, variance, etc.) and feed them into a simple ML classifier (SVM, random forest). This can quickly rank rules. Liao *et al.* essentially did this by combining entropy features with CNN-extracted features in their neural net [10].

- **Metric-based search:** Use heuristic metrics to guide a search or GA, then use ML to refine. For instance, one could use Langton's lambda (fraction of births) or density trends to narrow down to "edge-of-chaos" candidates, then apply a CNN classifier.

- **Predictive models:** Cisneros *et al.* (2019) propose using a **neural predictor**: train an RNN or MLP to predict the next CA state (or predict the complexity measure). They found that predictor networks, combined with compression metrics, can better identify rules that *continue* to create structure [5] [21]. For example, a network that learns to forecast patterns will struggle on random noise but do well on structured evolutions, so its prediction error can be an "interestingness" signal.

- **Ensembles of metrics:** Instead of picking one metric, build a feature vector of several: e.g. (time-averaged density, entropy, LZ-complexity, glider count). Then cluster or classify in this feature space. This hybridizes the strengths of each metric. Zenil showed that using both compression and conventional entropy gives a more complete picture of complexity [22].

- **Active learning/human-in-the-loop:** Use a human or visualization tool to label a few sample rules as interesting or not, train an ML model on those labels, and iteratively refine. Softology's blog suggests putting a "Look for interesting rules" button that runs automated filters and then lets the user pick from the outputs [23]. This semi-automated approach can greatly reduce the search burden.

In short, **hybrid methods** often outperform pure statistical or pure ML methods alone. For example, Cisneros *et al.* (2019) explicitly note that combining compression algorithms with neural network predictors yields metrics that can autonomously grow complexity over time [21]. Thus, a practical pipeline might first compute fast metrics to filter out ~90% of rules, then apply a trained CNN/RNN to the remaining ones, using the combination as a robust criterion.

6. Tools, Data and Visualization Aids

- **CA simulators:** *Golly* is a free, cross-platform CA explorer supporting Life-like rules; it allows scripting (Lua/Python) and is highly optimized for large grids. *Mathematica* has a built-in `CellularAutomaton` function and can simulate Life-like rules with code. The **CARLE** framework (PapersWithCode) provides a GPU-accelerated CA simulator covering all 262K life-like rules [20]. Using such tools can speed up batch simulations (CARLE claims "tens of thousands of steps/sec" across all rules [20]).

- **Known rule collections:** ConwayLife.com and related communities maintain lists of interesting Life-like rules (e.g. Day & Night, HighLife, Seeds variants, etc.). These repositories can seed training datasets. For example, Liao *et al.* started from known complex examples and performed data augmentation [10]. Similarly, Wikipedia's "Life-like cellular automaton" page lists many rule strings and known behaviors.

- **Visualization:** Tools for plotting space-time diagrams (like an image of successive grids) help human inspection. Some works use 3D "stack" visualizations (Wuensche's Fig.1 [6]). Automated visualizations (density vs time plots, Lyapunov exponent charts) can summarize behavior.

- **ML frameworks:** Standard deep learning libraries (TensorFlow, PyTorch) can be used for training CNN/RNN models on CA data. The Distill "Growing Neural CA" site [16] provides code and notebooks demonstrating CA learning with CNNs. The 3D-Growing-NCA GitHub repo [24] (from the Distill paper)

is another example implementation (albeit for generation, not analysis). These resources show how to encode CA states as tensors and train networks.

- **Datasets:** There is no widely adopted "Life-like CA dataset" labeled for interestingness, but one can create a dataset by sampling rules and labeling them via experts or metrics. However, note the existence of CARLE's environment (an open challenge) which provides code and a learning benchmark around life-like CAs [20]. For standard supervised tasks, one could use Rollier's labeled ECA classification data or generate synthetic labels from known classes.

In summary, the ecosystem includes high-performance simulators (Golly, CARLE), rule databases (ConwayLife, Wolfram demos), and ML code (neural CA libraries). Leveraging these can speed up development of an automated pipeline.

Summary of Recommendations

- **Use multiple metrics.** The two-dimensional "[density, complexity]" plot is a start, but supplement it. For each rule, compute *entropy*, *variance of cell-count*, *compressibility*, and *pattern counts* (gliders/oscillators if feasible). These complementary statistics better capture complex behaviors.
- **Apply deep models.** Train a CNN or hybrid model on space-time diagrams to classify interesting/boring. Use data augmentation and include both "positive" (known interesting rules) and "negative" examples. The literature shows CNNs can achieve ~80–90% accuracy [12] and even generalize to new rules [13].
- **Cluster smartly.** Represent each rule's output with features (e.g. learned CNN embeddings or autoencoder codes) and cluster. This can reveal outlier pockets of structure. In particular, try coarse-graining large simulations and then clustering (as in [19]) to filter out background randomness.
- **Optimize simulation.** Implement early-stopping for boring/chaotic outcomes, and vary initial seeds. Consider parallel acceleration (e.g. GPU via CARLE or bitwise methods). Run medium-length simulations (100–200 steps) on all rules first, then extend only the surviving subset to 1000+ steps.
- **Combine approaches.** A two-stage pipeline works well: first use fast heuristic filters (entropy threshold, simple ML classifier) to cut the rule space, then apply deeper analysis on the survivors. For example, use Shannon entropy to eliminate all-fixed rules, then feed the rest into a CNN trained on "glider presence". Such hybrids have been successful in open-ended evolution studies [21].
- **Use available tools.** Employ simulators like Golly or CARLE for speed. Leverage community code (e.g. Distill's neural CA examples [16]) to build/benchmark your models. Visualize clusters and CA patterns frequently to guide tuning.

By enriching the feature set beyond average density, using modern ML classifiers, and smart simulation strategies, one can much more effectively sift through the 2^18 life-like rules. Combining statistical measures with CNN/RNN analysis – as several recent papers demonstrate [10] [11] – offers the best chance of automatically identifying those rare rules that generate rich, life-like complexity.

References: (Key citations from the literature are given above in context. See e.g. Peña & Sayama (2021) [1], Liao (2021) [10] [12], Cisneros et al. (2019,2021) [21] [4], Zenil (2010) [3], Wuensche (1998) [2] [6], and others. These works discuss CA complexity metrics, ML approaches, and clustering techniques.】

---

1 **Life Worth Mentioning: Complexity in Life-Like Cellular Automata - PubMed**
https://pubmed.ncbi.nlm.nih.gov/34727158/

2 **sfi-edu.s3.amazonaws.com**
6 https://sfi-edu.s3.amazonaws.com/sfi-edu/production/uploads/sfi-com/dev/uploads/filer/fb/60/fb60632a-97fa-4080-9b21-769e0f65976d/98-02-018.pdf

3 **[0910.4042] Compression-based investigation of the dynamical properties of cellular automata and other systems**
https://arxiv.org/abs/0910.4042

4 5 21 **[1911.01086] Evolving Structures in Complex Systems**
22 https://ar5iv.org/pdf/1911.01086

7 23 **Automatic Detection of Interesting Cellular Automata | Softology's Blog**
https://softologyblog.wordpress.com/2019/09/03/automatic-detection-of-interesting-cellular-automata/

8 **Are there any symmetries in rules of Life-Like Cellular Automata? - Mathematics Stack Exchange**
https://math.stackexchange.com/questions/3470300/are-there-any-symmetries-in-rules-of-life-like-cellular-automata

9 **research.gold.ac.uk**
https://research.gold.ac.uk/17261/1/2014_TPNC_swarms_CA.pdf

10 12 **www2.eecs.berkeley.edu**
https://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-150.pdf

11 **[2409.02740] Convolutional Neural Networks for Automated Cellular Automaton Classification**
https://arxiv.org/abs/2409.02740

13 **juser.fz-juelich.de**
https://juser.fz-juelich.de/record/892991/files/Master_Thesis_MarcelAach.pdf?version=1

14 **A hybrid cellular automaton model integrated with 3DCNN and ...**
https://www.tandfonline.com/doi/full/10.1080/17538947.2024.2447337

15 **[PDF] Learning Graph Cellular Automata**
https://proceedings.neurips.cc/paper/2021/file/af87f7cdcda223c41c3f3ef05a3aaeea-Paper.pdf

16 **Growing Neural Cellular Automata**
https://distill.pub/2020/growing-ca/

17 **Differentiable Logic CA: from Game of Life to Pattern Generation**
https://google-research.github.io/self-organising-systems/difflogic-ca/

18 **[2104.01008] Visualizing computation in large-scale cellular automata**
https://ar5iv.org/pdf/2104.01008

19 **[2104.01008] Visualizing computation in large-scale cellular automata**
https://arxiv.org/abs/2104.01008

20 **CARLE Dataset | Papers With Code**
https://paperswithcode.com/dataset/carle

24 **GitHub - Aadityaza/3d-Growing-neural-cellular-automata: This project is inspired by the paper "Growing Neural Cellular Automata" found at the link: https://distill.pub/2020/growing-ca/. The project's objective is to develop an extension of the methodology described in this paper to enable the generation of complex structures in three dimensions.**
https://github.com/Aadityaza/3d-Growing-neural-cellular-automata