Graph-Augmented Neural Cellular Automata for Pattern Growth, Regeneration and Stability

Frederick Spreafichi

September 16, 2025

Contents

1 Introduction

Many natural and engineered systems exhibit *self-organization*: coherent global structure emerges from simple, local interactions. Recent work on *Neural Cellular Automata* (NCA) has shown that learned, strictly local update rules can grow, repair, and maintain complex target patterns from minimal seeds while remaining robust to perturbations (see the Distill thread on differentiable self-organizing systems Distill 2020 and the article Growing Neural Cellular Automata). In this setting, each cell stores a small vector state, senses its neighborhood through a fixed "perception" stack, and applies a tiny neural network to produce a bounded state update. When trained end-to-end, the resulting dynamics can make a desired pattern a stable attractor: the system grows the target from a seed, repairs after damage, and then holds steady.

A key limitation of classic grid NCAs is *range*: information propagates by repeated local steps (e.g., 3×3 neighborhoods). While this preserves cellularity and locality, certain global coordination tasks may benefit from *mid-range* interactions. Our goal in this work is to retain the spirit of NCA (locality, bounded updates, asynchronous robustness) while adding a carefully controlled, sparse mechanism for mid-range communication.

Approach. We use as a basis a standard Neural Cellular Automaton that we enrich by implementing an attention-weighted mid-range message passing computed over a small, randomly sampled set of spatial offsets outside the classical local 3×3 neighborhood of the NCA. At each step the model:

- computes the usual update field from a local, fixed, depthwise perception (identity + Sobel filters),
- forms a sparse, attention-weighted aggregation of features shifted by the sampled offset (this is meant to mimic the graph neural network information propagation behavior).

The message-passing phase operates on the hidden channels of the cell's state vector, is bounded using a *tanh*, and scaled before being injected as a residual into the update state for the cells. Alive gating alongside an alpha-only post gate keeps the growth confined and the stochastic fire rate serves as an asynchronous regularizer. The result is a bounded, gated, cellular update rule that also utilizes sparse mid-range coordination via an attention-weighted graph message passing.

Contributions.

- **Graph-augmented NCA (GNCA).** A transition rule that combines fixed local perception, a 1×1 update MLP (zero-initialized last layer), GroupNorm on the update field, and a bounded mid-range attention message added as a residual prior to normalization.
- **Safe routing and gating.** Hidden-only message routing, alive $\rightarrow$ alive messaging, pre-update alive gating, and alpha-only post gating jointly promote stability and a richer intra-network communication (local and mid-range), while keeping the evolution of the NCA smooth.
- **Empirical evaluation.** We assess growth from seed, regeneration after structured damages, and steady-state maintenance, reporting pixel-wise accuracy and perceptual metrics.

Scope. We focus on single-target pattern growth on 2D canvases with toroidal boundaries (roll-based shifts) and evaluate whether mid-range attention improves coordination without sacrificing cellular stability. The architecture is theory-first and intentionally minimal: we avoid global convolutions, normalize only the update field, and strictly bound each step.

2 Graph-Augmented Neural Cellular Automata

We build on the classic Neural Cellular Automaton design described in Growing Neural Cellular Automata. The base NCA rule uses a fixed local perception (identity and Sobel filters, for edge detection), a pointwise neural update (two 1×1 convolutions), and a bounded residual step. On top of this, we introduce a sparse, attention-weighted mid-range message that enables coordination beyond the local 3×3 neighborhood while still preserving locality and stability. The idea is that by enabling sparse mid-range information to be propagated within the network, we will obtain a more stable and richer architecture (in terms of context awareness of the whole network).

State layout. We model the canvas as a tensor $x_t \in \mathbb{R}^{C \times H \times W}$ with C channels (RGBA+hidden): channels 0–2 are RGB, channel 3 is the alpha occupancy field $\alpha(x_t)$, and channels $4 : C - 1$ are hidden. One CA step produces x_{t+1} from x_t. The training target is a fixed image $\tau \in [0, 1]^{4 \times H \times W}$ on the visible channels. (For clarity we omit the batch dimension in notation; the implementation operates on $x_t \in \mathbb{R}^{B \times C \times H \times W}$).

2.1 NCA base

Dilated aliveness. A cell is *alive* if its alpha, after 3×3 max-pooling, exceeds a threshold μ:

$$A(x_t) \;=\; \mathbf{1}\big[\mathrm{MaxPool}_{3\times3}\big(\alpha(x_t)\big) > \mu\big] \in \{0,1\}^{1\times H\times W}.$$

The 3×3 max-pool acts as a one-pixel dilation of the alpha support. Dilation lets growth occur at the *frontier* of existing tissue (new cells update next to living ones) while preventing distant, isolated pixels from "sparking" alive in empty space. This yields boundary-led growth and suppresses background sprouts.

Where the mask is used. (i) **Pre-update gate:** the update field is applied only where $A(x_t) = 1$. (ii) **Post-update gate (alpha only):** after applying the step, the new alpha is gated as

$$\alpha(x_{t+1}) \leftarrow \alpha(\tilde{x}_{t+1}) \odot A(\tilde{x}_{t+1}).$$

The pre-gate localizes computation to living tissue and its immediate boundary; the alpha-only post-gate guarantees that only sufficiently strong alpha persists, removing weak, noisy activations while allowing RGB/hidden to continue adapting.

Fixed depthwise perception. A frozen depthwise operator $P(\cdot)$ applies identity, Sobel-x, and Sobel-y to each channel:

$$y_t = P(x_t) \in \mathbb{R}^{3C\times H\times W}.$$

Freezing identity+Sobel injects stable, derivative-like cues from the start, reduces parameters, and improves optimization stability. Since the loss supervises the final image, learnable first-layer filters would otherwise drift away from clean edge detectors to whatever reduces loss; keeping them fixed makes the update rule learn how to use edges/gradients rather than re-learn them.

Pointwise update MLP. A channel-mixing 1×1 MLP produces the *raw* local update:

$$d_t^{\mathrm{loc}} \;=\; U(y_t) \;=\; W_2\,\phi(W_1 y_t + b_1) \;\in\; \mathbb{R}^{C\times H\times W}, \qquad \phi = \mathrm{ReLU}.$$

The output layer W_2 is *zero-initialized* so dynamics start quiescent. We intentionally leave d_t^{loc} unnormalized here; normalization and step bounding are applied *once*, after combining with the mid-range message and applying the alive/firing masks (see "Finalize the step").

2.2 Mid-range graph message passing

Offsets. To coordinate beyond the 3×3 neighborhood, we add a sparse mid-range message. Let

$$\Omega \;=\; \{(dx,dy) : \max(|dx|,|dy|) \le r\} \setminus \{(dx,dy) : |dx| \le 1, |dy| \le 1\}$$

be the square stencil of radius r with local neighbors removed. At each step we sample K offsets $\{o_k\}_{k=1}^{K} \subset \Omega$.

Keeping $K \ll |\Omega|$ preserves the cellular character (local behavior), avoids a constant global push, promotes directional isotropy over time (no single direction is privileged), and keeps compute overhead small.

Boundary handling. We shift features by $o = (dx, dy)$ via either:

- **Toroidal wrap** $\mathrm{roll}(\cdot, o)$ (indices modulo H, W): values leaving one side re-enter from the opposite side.
- **Zero padding**: out-of-domain values are zeros (hard boundaries).

We use *toroidal wrap* by default.

Roll preserves translation invariance, avoids low-signal border bands that bias growth, and keeps each cell's mid-range degree uniform across the canvas. Zero padding is a drop-in alternative when hard edges are semantically important.

Projections and semantics. We employ a lightweight, transformer-style attention with 1×1 projections:

$$Q = W_Q x_t, \quad K = W_K x_t \ \in \ \mathbb{R}^{d\times H\times W}, \qquad M = W_M x_t \ \in \ \mathbb{R}^{C\times H\times W}.$$

Interpretation: Q encodes *what a receiver needs*, K encodes *what a sender offers*, M is the *content* to be sent.

This factorization is simple, interpretable, and cheap: 1×1 projections keep per-step cost linear in CHW.

Affinity and aggregation. We pool queries and shifted keys to obtain *per-step, per-sample* weights over offsets:

$$\bar{q} \ = \ \mathrm{mean}_{h,w}(Q), \qquad \bar{k}^{(o)} \ = \ \mathrm{mean}_{h,w}\big(\mathrm{roll}(K, o)\big).$$

With a learnable temperature $\beta > 0$ we form logits

$$a^{(o)} \ = \ \frac{\langle \bar{q}, \bar{k}^{(o)} \rangle}{\beta}, \qquad w^{(o)} \ = \ \frac{\exp(a^{(o)})}{\sum_{o' \in \{o_k\}} \exp(a^{(o')})},$$

and aggregate *spatial* messages from shifted content maps:

$$m_t \ = \ \sum_{o \in \{o_k\}} w^{(o)} \ \mathrm{roll}(M, o) \ \in \ \mathbb{R}^{C\times H\times W}.$$

The vector $w^{(o)}$ is a probability distribution over the sampled offsets: it selects which directions/distances (the mid-range links) should carry information at this step. The same $w^{(o)}$ applies to all pixels, but $\mathrm{roll}(M, o)$ is a full image-sized tensor, so m_t remains spatially structured.

Pooling avoids the $\mathcal{O}(HW)$ cost of per-pixel attention, keeps the mechanism easy to interpret, and helps preserve the cellular character: we coordinate *which offsets* to use, not perform dense nonlocal mixing.

Smaller β (low temperature) makes $w^{(o)}$ peaky, approaching a hard choice of a few offsets; larger β averages across offsets. This lets training adapt selectivity: sharp routing during growth/repair, smoother averaging near steady state.

Alive→Alive and routing. We restrict senders to living tissue via $\mathrm{roll}(M, o) \odot \mathrm{roll}(A(x_t), o)$. We apply a *hidden-only* policy by zeroing $(m_t)_{0:4}$ so the graph modulates hidden dynamics rather than directly painting RGB/alpha.

Tissue-to-tissue communication reduces background artifacts; hidden-only routing preserves the causal pathway "hidden dynamics → visible state", improving stability and similarity to organic systems.

(An optional per-channel gate $g_t = \sigma(G([x_t; m_t]))$ is implemented but disabled in our experiments to keep the mechanism minimal.)

Bounded residual into the update. We bound and scale the message, then inject it at the update level:

$$m_t^\star \;=\; \gamma\,\tanh(m_t), \qquad d_t \;=\; d_t^{\mathrm{loc}} + m_t^\star.$$

Residual updates have several advantages over multiplicative updates for GNCA:

- **Guaranteed, state-independent step bound.** With residuals, each component of the step is absolutely bounded by the gain and tanh (after group norm (GN)). In multiplicative form the effective step size scales with $|x_t|$, so large states yield large (potentially destabilizing) jumps, while tiny states barely change.
- **Can create signal from (near) zero.** Growth from seeds requires adding new mass at the frontier. If a channel at some site is zero (or very small), multiplicative updates keep it at zero (or near zero); residual updates can raise it above threshold and start new tissue at the boundary.
- **Better fixed points and gradient flow.** Residual dynamics keep the identity map available ($d_t \approx 0 \Rightarrow x_{t+1} \approx x_t$), which eases optimization. Multiplicative chains compound, causing vanishing/exploding behaviors and making stable attractors harder to learn.
- **Co-normalization with the local rule.** By adding $m_t^\star$ *before* $\mathrm{GN} \to \tanh$, the local update and the mid-range message are normalized and step-bounded *together*. A multiplicative path would bypass this shared bounding unless one also re-normalizes x_t itself, which harms memory.
- **Cleaner interaction with alive gates.** The pre-update gate masks the update field d_t; with residuals this directly controls where the state can change. In multiplicative form, even small s_t can amplify/suppress existing values in alive regions in a state-dependent way, and cannot *initiate* new alpha where it is currently zero, contrary to our growth-from-frontier objective.

For these reasons we keep the graph term as a *bounded residual into the update* rather than a multiplicative modulation of the state.

Adding the graph signal *before* $\mathrm{GN} \to \tanh$ means local and mid-range contributions are co-normalized and step-bounded together, preventing the message from destabilizing the dynamics.

2.3 Stochastic firing, normalization, and post gate

Asynchrony (fire-rate). At each inner step we sample a fire-rate $p \in [p_{\min}, p_{\max}]$ and draw an i.i.d. Bernoulli mask $B_t \in \{0,1\}^{1\times H\times W}$ (shared across channels) with $\mathbb{P}[B_t(h,w)=1] = p$. Only fired sites update on that step; others hold their state:

$$d_t \leftarrow d_t \odot B_t \odot A(x_t).$$

Thus, a fraction $\approx p$ of cells applies the rule per step, mimicking asynchronous cellular automata. This reduces reliance on strict synchrony, improves robustness to update order, and acts as a regularizer. (During training, p can be varied per step within a range to expose the dynamics to different levels of sparsity.)

Finalize the step. Normalize and bound the combined update, apply it, and post-gate alpha:

$$\tilde{d}_t \;=\; \eta\,\tanh\big(\mathrm{GN}(d_t)\big), \qquad \tilde{x}_{t+1} \;=\; x_t + \tilde{d}_t, \qquad \alpha(x_{t+1}) \;=\; \alpha(\tilde{x}_{t+1}) \odot A(\tilde{x}_{t+1}).$$

GN on the *update* stabilizes per-step magnitudes even with sparse alive regions and small batches; the post alpha-gate prevents leakage while RGB/hidden remain free to fine-tune.

2.4 One-step summary

$$\boxed{\begin{aligned}
&y_t = P(x_t), \quad d_t^{\text{loc}} = U(y_t), \quad m_t = \sum_{o \in \{o_k\}} w^{(o)} \, \text{roll}(W_M x_t, o), \\
&d_t = d_t^{\text{loc}} + \gamma \, \tanh(m_t), \quad d_t \leftarrow d_t \odot B_t \odot A(x_t), \\
&\tilde{d}_t = \eta \, \tanh(\text{GN}(d_t)), \quad \tilde{x}_{t+1} = x_t + \tilde{d}_t, \quad \alpha(x_{t+1}) = \alpha(\tilde{x}_{t+1}) \odot A(\tilde{x}_{t+1}).
\end{aligned}}$$

Stability. Zero-initialized W_2 (quiescent start), GN $\to \tanh \to$ small gain on the *update*, hidden-only bounded message, and pre/post alive gates jointly produce small, well-scaled steps and robust long-horizon rollouts.

Range vs. locality. The mid-range message is *sparse* (few offsets per step) and attention-weighted, so it augments rather than overrides local interactions — improving coordination (growth, repair) without losing the cellular character.

3 Loss

Premultiplied RGBA objective. We supervise the visible channels with a full-canvas MSE on *premultiplied* RGBA, following *Growing Neural Cellular Automata*. Let x_T be the terminal CA state (first four channels are RGBA) and $\tau \in [0,1]^{4\times H\times W}$ the target. Define

$$\text{prem}\left(\begin{bmatrix}\mathbf{r}\\ \alpha\end{bmatrix}\right) = \begin{bmatrix}\alpha\,\mathbf{r}\\ \alpha\end{bmatrix}, \qquad \mathbf{r} \in [0,1]^{3\times H\times W}, \ \alpha \in [0,1]^{1\times H\times W}.$$

We set $\hat{y} = \text{prem}\big(x_T^{\text{RGBA}}\big)$ and $\tau^\star = \text{prem}(\tau)$ and minimize

$$\mathcal{L}_{\text{RGBA}} = \frac{1}{4HW}\big\|\hat{y} - \tau^\star\big\|_2^2 = \frac{1}{4HW}\sum_{c,h,w}\big(\hat{y}_{c,h,w} - \tau^\star_{c,h,w}\big)^2.$$

In practice, τ is premultiplied once on load, while the prediction is premultiplied on-the-fly before the loss.

Rationale. Premultiplication ties RGB errors to the alpha support: background pixels (α=0) contribute no RGB loss, while any *alpha overshoot* is penalized directly via the alpha channel. This removes pressure to "paint" the background and empirically suppresses the overgrow-then-carve failure mode without extra background penalties or masks.

Supervision timing. For each minibatch sample we draw a rollout length (short/long curriculum), unroll the CA with stochastic firing and sparse mid-range messages, and compute $\mathcal{L}_{\text{RGBA}}$ *once* on the terminal state. No intermediate-step supervision is used.

3.1 Training protocol

Seeds and symmetry breaking. We initialize x_0 as all zeros except for a single central cell with $\alpha = 1$; hidden channels at that cell receive small Gaussian noise ($\sim 0.01\mathcal{N}(0,1)$). This breaks symmetry and provides a minimal growth nucleus.

Population (pool) training. We maintain a pool of states and draw minibatches from it. After each update we return the evolved states to the pool, but (i) replace a small fraction of the worst samples (highest per-sample loss) with fresh seeds, and (ii) occasionally reseed a random sample. This maintains diversity over rollout times and damage conditions and prevents mode collapse to "easy" trajectories.

Damage curriculum (regeneration). Before each rollout we apply a stochastic damage operator $D_{\text{epoch}}(\cdot)$ to the current state (random masks/holes/strokes per the configuration). Training on damaged states teaches the automaton to repair, not just grow.

Rollout schedule (short/long) and asynchrony. For each sample we draw a rollout length T from a short range with high probability and from a long range otherwise. Within the rollout, at each step we draw a fire-rate $p \in [p_{\min}, p_{\max}]$ and update only the fired sites (asynchronous CA). This exposes the rule to both early growth and long-horizon maintenance under stochastic update order.

Temporal sparsity and warm-up of mid-range messages. At step t, the mid-range message is applied only on a subset of steps (via a rate `message_rate` or period `message_every`). In addition, the message gain γ is warmed up across epochs (small at first, then increased). This lets the local rule form a stable backbone before the graph term provides coordination.

Optimization details. We optimize with Adam (weight decay as configured), clip gradients to 0.5, and optionally use a learning-rate scheduler (StepLR or cosine). These choices improve stability with sparse alive regions and long rollouts.

Supervision timing. For each minibatch we unroll the CA for the sampled number of steps and compute the loss *once* on the terminal visible state (no intermediate-step supervision). For samples below a stability threshold, we run K additional steps and add a stability MSE on their new terminal states; the batch loss averages per-sample totals.

4 Results

We trained on an emoji dataset for a total of **960 epochs** on a **40×40** pixel canvas (each pixel is a cell), using population (pool) training with stochastic damage, asynchronous firing, and temporally sparse mid-range messaging.

4.1 Metrics

Pixel perfection (PP, ideal 1.0). Fraction of pixels whose *per-channel* absolute error on RGBA is below a fixed threshold ε. Concretely, with $\varepsilon = 0.05$ we count a pixel (h, w) as correct if $\max_{c \in \{\text{R,G,B},\alpha\}} |\hat{y}_{c,h,w} - \tau_{c,h,w}| < \varepsilon$, then average over the image. Evaluated on the *terminal* rollout state; predictions are clipped to $[0, 1]$ before scoring (ideal range: $[0, 1]$, best $= 1$).

SSIM on RGB (ideal 1.0). Structural Similarity Index computed on the *RGB* channels of the terminal state with data range 1.0 (standard implementation; higher is better). Captures contrast, luminance, and structural agreement (typical range $[0, 1]$, best $= 1$).

PSNR on RGB (dB, ideal $+\infty$). Peak Signal-to-Noise Ratio in decibels on the *RGB* channels of the terminal state with peak value 1: $\text{PSNR} = 10 \log_{10}(1/\text{MSE})$ (higher is better). Typical trained models fall in $\sim$20–50 dB; ideal is unbounded above.

4.2 Quantitative outcomes

Metric	Best (epoch)	Mean over run
Loss ↓	**0.001267** (99)	0.015351
PP ↑	**0.8377** (97)	0.5986
SSIM ↑	**0.9344** (943)	0.8207
PSNR (dB) ↑	**32.53** (99)	25.96

Table 1: Summary from the training log (epochs 1–961). PP: Pixel Perfection; SSIM/PSNR computed on RGB; full-canvas premultiplied-RGBA supervision.

Learning dynamics. Peak *PSNR* and lowest *loss* occur early (around epoch 100), while *SSIM* continues to improve much later (best at epoch 943), consistent with late-stage structural refinement. *Pixel Perfection* peaks at epoch 97 (0.8377) and is more sensitive to small boundary/alpha discrepancies than SSIM/PSNR, explaining its earlier peak relative to SSIM. Overall means (*PP* 0.5986, *SSIM* 0.8207, *PSNR* 25.96 dB, loss 0.015351) indicate strong perceptual fidelity and stable long-horizon behavior.

Parameters count. The GNCA remains lightweight: from the final checkpoints we measured **11,185** total parameters (*10,753* trainable) for the graph-augmented model versus **8,784** for the classic NCA. The graph augmentation therefore adds only **2,401** parameters (about **27%** relative increase) while enabling mid-range coordination.

5 Discussion: Negative Results and Future Directions

5.1 Failed attempts

Penalty-based masked losses (carving persists). We tried a masked reconstruction loss on target support plus background α/RGB and area penalties. Despite careful tuning, the method was brittle: it required a threshold and three coefficients, traded off foreground fidelity against background cleanliness, and frequently produced the "overgrow then carve" behavior (thin halos and ragged boundaries). Switching to a *premultiplied* RGBA loss removed the need for hand-tuned background terms and eliminated carving.

Updating the *state* multiplicatively instead of adding a residual (unstable). Direct state modulation ($x \leftarrow x \odot (1 + s)$) led to scale-dependent steps, poor gradient flow, and inability to create mass from near-zero states at the growth frontier. Training became sensitive to learning rate and often diverged or converged to fragile limit cycles. Residual updates ($x \leftarrow x + d$) with shared GN$\rightarrow$ tanh bounding were consistently stable.

Dropping the zero-init on the last layer (chaotic starts). Without zero-initializing W_2, early steps produced large, unstructured updates that overwhelmed the alive/post gates. Training required much smaller learning rates and still exhibited noisy textures and drift. Zero-init yielded a quiescent start and smoother learning.

No gradient clipping (exploding updates on long rollouts). With asynchronous firing, occasional saturated tanh regions and sparse alive masks produced rare but large gradients; without clipping these spikes caused instability or NaNs during long rollouts. A small clip (0.5) removed these failures without slowing convergence.

LayerNorm instead of GroupNorm(1,C) on the *update* (mismatch with masks). LayerNorm normalizes across (C, H, W) per sample; when the alive region is small, its statistics are dominated by zeros/background, leading to over-normalization and washed-out updates. GN$(1, C)$ normalizes per-site across channels, decoupling update scale from alive area and interacting cleanly with pre-/post-gates.

5.2 Future directions

Longer training. Train for a longer number of epochs, ideally more than 2000, with appropriate fine-tuning in order to improve performance and stability.

Richer targets. Train on more complex, high-frequency images (fine textures, multi-part shapes, varying alphas) to test the limits of coordination and steady-state fidelity.

Graph sparsity & weighting. Systematically sweep the mid-range design: radius r, sampled neighbors K, temperature β, message gain γ, and temporal sparsity (rate/period). Explore learned offset distributions or annealed/Gumbel-softmax routing to sharpen/discretize links as training progresses.

Longer rollouts and stability stress-tests. Evaluate stability over very long horizons (e.g., 10^3–10^4 steps), under repeated or persistent damage, and across a range of fire-rates. Look for drift, limit cycles, and basin-of-attraction size.

3D extension. Lift GNCA to volumetric grids ($C \times D \times H \times W$) with 3D Sobel perception and 3D offset stencils; assess growth/repair of volumetric shapes and the effect of sparse mid-range links in 3D. Memory and compute will require careful sparsity and checkpointing.