Using Different PDE Solvers for Noise-NCA-Based Texture Synthesis

Xinran Li†, Siyuan Zhang†
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland
{xinran.li, siyuan.zhang}@epfl.ch
† Authors contributed equally and ordered alphabetically.

***Abstract*—Neural Cellular Automata (NCA) is a class of Cellular Automata whose update rules are described by trainable neural networks and are generally considered to be an approximation of a continuous PDE. We investigated texture synthesis using Noise-NCA and explored the impact of different PDE solver combinations on its performance. Our results show that when training with low-order numerical methods, Noise-NCA cannot approximate the continuous PDE well, but has obvious overfitting. On the other hand, the use of higher-order method training may help to enhance the generalization ability of Noise-NCA at different time steps.**

I. Introduction

Cellular Automata (CA)[1], as a typical class of dynamical systems, exhibits discrete characteristics in space, time, and state. The characteristics of CA make them highly suitable for describing various phenomena in the field of imaging. Building on this foundation, Neural Cellular Automata (NCA)[2][3] utilizes neural networks to learn the dynamics encoded in data, achieving widespread application in computer graphics.

The Noise-NCA model[4] treats the target video as a second-order continuous differential equation for the original image, learning its continuous dynamic information. To make the final training model more adaptable in various temporal and spatial differences, Noise-NCA removes the stochastic updates from the NCA architecture and initializes cell states with random uniform noise, ultimately achieving good results in in Multiscale Patterns, Anisotropic Scaling, and Time-Varying Scaling.

Inspired by the reaction-diffusion systems described by Partial Differential Equations (PDEs)[5] (Turing, 1990), the update rule of Noise-NCA can be interpreted as an Euler integration of a PDE over a discretization of space and time. To further verify whether the Noise-NCA model has learned the structure of the PDE or is overfitting due to discretization used in the Euler Method, this article selects different numerical solutions for the PDEs as the update strategies for forward during the training and testing phases. Our final results show that even for the Noise-NCA model, overfitting exists. In addition, we also discover that the use of a higher-order training solver can help improve the generalization ability of the Noise-NCA model.

II. Related Works

Mordvinstev et al.[3] introduce the NCA model and demonstrate its capability in developing diverse self-organizing systems. Niklasson et al.[6] use the NCA model for texture synthesis for the first time and systematically evaluated its performance. To generate real-time controllable dynamic texture videos, the DyNCA model[7] integrates multi-scale perception and positional encoding on top of the NCA model, using the proposed appearance loss and motion loss. The cells in DyNCA can easily perform long-range communication and gather global information, ensuring enhanced performance both in terms of visual quality and computational expressivity.

To further improve the stability of the model, especially for better video generation results under variable scales, the Noise-NCA model[4] sets the initial condition to random noise and removes the need for stochastic updates in the update rule. The model has demonstrated better performance than previous work, and the authors claim that the method can be effectively fitted to a continuous PDE by adding noise.

III. Method

A. Concepts and Definitions

The NCA model processes images (textures) in pixels. For a pixel at (x, y), the state of the pixel is denoted as $S_{(x,y)}(t)$ at time t. $S_{(x,y)}(t)$ is encoded as a vector of length L. In the NCA update rules, $S_{(x,y)}(t)$ is updated iteratively over time steps and eventually converges to our desired state. At this point, we get a full, high-quality texture.

B. NCA Architecture

The network structure of NCA can be summarized into two parts, the first part is called perception and the second part is called adaptation. For each pixel, the output of these two parts is the amount of change in $S_{(x,y)}(t)$, denoted as $\Delta S_{(x,y)}(t)$. Figure 1 depicts the process of updating a pixel.

In the perception phase, for each pixel, Identity (0th order), Sobel-X, Sobel-Y (1st order) and Laplacian filters (2nd order) are used in the deep convolutional layer to extract their surrounding features and obtain intermediate outputs

$$InS_{(x,y)}(t) = Conv(S_{(x,y)}(t), [F_{ID}, F_X, F_Y, F_{Lap}])$$

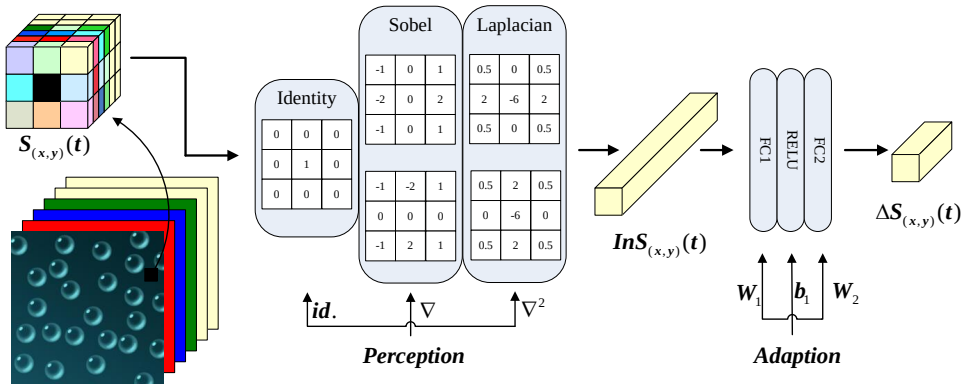

Figure 1: Illustration of a single step in NCA models.

In the adaptation phase, the intermediate results are fed as input to a two-layer neural network that uses ReLU as the activation function, with each of the layer having a weight W_1 and a weight W_2, but only the first layer has a bias b.

Considering that the nonlinear transformation represented by the neural network is f_n, then the final output can be expressed as

$$\Delta S_{(x,y)}(t) = f_n(InS_{(x,y)}(t))$$

C. Update as PDE

From a mathematical point of view, the perception and adaptation phase of NCA is actually simulating a PDE.

$$\frac{\partial S_{(x,y)}(t)}{\partial t} = f(S_{(x,y)}(t), \nabla_x S_{(x,y)}(t), \nabla_y S_{(x,y)}(t), \nabla^2 S_{(x,y)}(t))$$

In the perception phase, the model uses filters to approximate the spatial gradient of the corresponding order on discrete space. In the adaptation phase, the model learns a nonlinear transformation through f_n neural network to approximate f.

Now, in order to effectively update to the next state of the pixel $S_{(x,y)}(t+1)$, NCA can employ various numerical integration methods as update rules.

We have selected three classical numerical methods, which are described in Table I. Here, for NCA, $t_n = t$, $h = \Delta t$, $y_n = S_{(x,y)}(t)$, and $y_{n+1} = S_{(x,y)}(t+1)$. With these numerical methods, we update the state of the pixels and, over time, the whole image eventually converges to a high-quality texture.

During the training and testing phases of NCA, we can update the state of the pixels using different numerical methods. The numerical method chosen during the training phase may affect the generalization ability of the resulting model. This is because it is possible for the model to overfit the discrete-time patterns of the state of some numerical method update than to learn about continuous PDEs. The more severe this overfitting, the worse the generalization ability of the model at different time steps. Besides, the numerical method chosen during the testing phase affects the ability and quality of the textures to be generated. If a method learns continuous PDEs during the training phase, using a higher-order numerical method will have a smaller global truncation error and higher accuracy during the testing phase, so the resulting texture should be of higher quality.

To synthesize textures at varying scales and speeds, the ideal state of NCA is to fit to a continuous PDE, rather than overfitting to a discrete update pattern of a numerical method in training. However, the previous work[3][7][4] was only updated using the Euler method, and did not consider that different solvers can be used in the training and testing phases, so there is a lack of analysis of the relationship between the type of PDE solver and the generalization ability of the model. In the experimental part, we focus on the selection and combination of solvers and analyze the performance of the corresponding models. Our experiments are based on Noise-NCA, as other models have been shown

Table I: Numerical methods and their update rules.

Numerical Method	Update Rule
Euler Method	$y_{n+1} = y_n + hf(t_n, y_n)$
Improved Euler Method (Heun's Method)	$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_n + h, y_n + hf(t_n, y_n))]$
Fourth-Order Runge-Kutta Method	$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$, $k_1 = f(t_n, y_n)$, $k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$, $k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$, $k_4 = f(t_n + h, y_n + hk_3)$

to be incapable of generalizing to different time steps[4].

IV. Experiments and Analysis

A combination of different numerical methods (to simplify, we call them solvers) was considered during the training and testing phases. We used the Euler method (Euler), the improved Euler method (ImE), and the fourth-order Runge-kuta method (RK4) as the solvers. In the training stage, we chose epoch = 3000, $\Delta t = 1.0$, and we used the Adam optimizer with an initial learning rate of 1e-3 and adopted a multi-step learning rate scheduler to accelerate convergence. At the 1000th and 2000th steps, the learning rate decays to 30% of its previous value. The pool technique proposed by Mordvinstev et al.[8] was used to suppress overfitting, the pool size was 256, the number of samples per time was 4, and a noise seed was added to the pool every 20 epochs.

During the testing phase, we used each of these three solvers to generate textures separately. To obtain a reliable model results that either converge to a given texture or diverge, which represents that the model cannot produce given texture and may cause overflow or underflow issues, the total time tested was 1000 time units for all time step settings. Considering that Noise-NCA generates textures at different locations over time steps, we used the average of the losses of the last 300 time units as the loss function values for each run. Each data point was the average of 10 independent runs.

Due to the time and computational resource constraints of the project, we selected 20 images from the 45 images in the original dataset[7] for our experiment. These images were selected from different types of textures to ensure that the results are sufficiently representative.

A. Varying Time Steps: Basic Analysis

We tested with the varying time step sizes: $\Delta t \in \{1.0, 0.5, 0.1\}$ and the convergence/divergence of the 9 solver combinations on all 20 images is shown in Table II. The convergence ratio was the ratio of the number of textures that a model could successfully generate (i.e., the loss function convergence) to the total number of textures in the dataset.

In the experimental results, it is reasonable that the higher-order testing solvers have better convergence than the lower-order testing solvers because they have smaller global truncation errors. In the same way, it is reasonable to have better convergence results with smaller time steps. However, let's consider the assumption that if Noise-NCA learns a continuous PDE instead of overfitting the numerical method at a discrete time step, then for the same time step Δt, the training solver of different orders should learn the same PDE, i.e., there should be $F_{\Delta t/Euler} = F_{\Delta t/ImE} = F_{\Delta t/RK4}$, where F represents the final output. But in fact, when $\Delta t = 1.0$, the Euler method testing solver always converged on $F_{\Delta t/Euler}$, while diverging on $F_{\Delta t/ImE}$ and $F_{\Delta t/RK4}$; When solving $F_{\Delta t/Euler}$ and $F_{\Delta t/ImE}$, ImE testing solver always converged, while it diverged on solving $F_{\Delta t/RK4}$. Even for $\Delta t = 0.5$, using the Euler method as the test solver to solve $F_{\Delta t/ImE}$ or $F_{\Delta t/RK4}$ still did not converge well. This suggests that $F_{\Delta t/Euler} \neq F_{\Delta t/ImE} \neq F_{\Delta t/RK4}$. Otherwise, for the same time step, the same testing solver should exhibit the same convergence/divergence behavior for the model trained with different training solvers. Therefore, Noise-NCA still overfits the numerical method at discrete time steps. We also give an intuitive example of texture generation to show the visual effect of this overfitting on texture quality in Figure 2 and the loss values in Table III.

Table II: Convergence ratio for different training and testing solvers with varying time steps.

Training-Testing Solver Combinations	Time Step Size		
	1.0	**0.5**	**0.1**
Euler-Euler	1.00	1.00	1.00
ImE-Euler	0.00	0.45	1.00
RK4-Euler	0.00	0.25	1.00
Euler-ImE	1.00	1.00	1.00
ImE-ImE	1.00	1.00	1.00
RK4-ImE	0.35	1.00	1.00
Euler-RK4	1.00	1.00	1.00
ImE-RK4	1.00	1.00	1.00
RK4-RK4	1.00	1.00	1.00

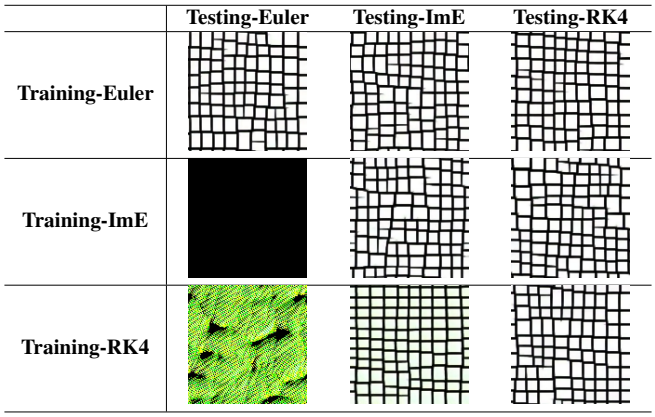

Figure 2: The visual effect of overfitting on texture quality when $\Delta t = 1.0$.

Table III: Corresponding loss values of Figure 2.

Training Solver	Loss of Testing Solver		
	Euler	**ImE**	**RK4**
Euler	6.8633	6.3258	6.3158
ImE	INF (Diverge)	6.7215	6.3531
RK4	INF (Diverge)	10.5031	6.1457

Table IV: The Wilcoxon signed-rank test results for pairing different training solvers.

Testing Solver	Results of Training Solver Pairs		
	(Euler, ImE)	**(Euler, RK4)**	**(ImE, RK4)**
Euler	$L_{ImE} < L_{Euler}$, p = 0.0484	$L_{RK4} < L_{Euler}$, p = 0.0055	Not Significant, p = 0.4304
ImE	$L_{ImE} < L_{Euler}$, p = 0.0056	$L_{RK4} < L_{Euler}$, p = 0.0094	Not Significant, p = 0.1327
RK4	$L_{ImE} < L_{Euler}$, p = 0.0032	$L_{RK4} < L_{Euler}$, p = 0.0153	Not Significant, p = 0.8695

Table V: The Wilcoxon signed-rank test results for pairing different testing solvers.

Training Solver	Results of Testing Solver Pairs		
	(Euler, ImE)	**(Euler, RK4)**	**(ImE, RK4)**
Euler	$L_{Euler} < L_{ImE}$, p = 0.0014	Not Significant, p = 0.1054	$L_{RK4} < L_{ImE}$, p = 0.0362
ImE	Not Significant, p = 0.2024	Not Significant, p = 0.9273	$L_{RK4} < L_{ImE}$, p = 0.0083
RK4	Not Significant, p = 0.6742	Not Significant, p = 0.8694	Not Significant, p = 0.4980

B. Further Analysis: Matched-Pairs Test

When $\Delta t = 0.1$, all testing solvers converged on all models for all images. We measured the quality of the texture using the loss function value at testing time and compared the performance of the model when using different training and testing solvers. However, performance comparisons may vary for different images. In order to obtain statistically significant results, we paired the values of the loss function obtained under different configurations and performed the Wilcoxon signed-rank test. The reason for not using the paired t-test was that we could not guarantee that the difference of the paired samples follows normal distributions[1]. For different testing solvers, the results of pairing the training solvers are shown in Table IV. For different training solvers, the results of pairing the testing solvers are shown in Table V.

For Table IV, we find that the loss value of the model using the Euler method as the training solver is significantly greater than that of the other two models, regardless of the test solver used. This means that it exhibits the worst generalization ability, with significant overfitting. The quality of the textures generated by the models trained using the improved Euler and the fourth order Runge Kuta methods is not much different.

For Table V, when the training solver is the Euler method, the testing solver performance using Euler's method and the fourth-order Runge Kuta method is significantly better than that of the solver using the improved Euler method. However, recall that we asserted in our analysis of Table IV that the model itself is heavily overfitting when training with the solver using the Euler method, so this performance gain is not of much significance, but may only be a side effect of overfitting (especially for testing solvers using the Euler method). When the training solver uses the other two methods, most of the difference in performance between the testing solvers is not statistically significant. Overall, the fourth-order Runge-Kuta method performed slightly better, but it also ran slower.

When the step size changes, what is the changing trend of the loss value? We only consider those combinations that use the same training solver and test solver (since they converge for all $\Delta t \in \{1.0, 0.5, 0.1\}$ well), and test the relationship between their loss values under different step sizes. The results are shown in Table VI.

When using the Euler method as the solver, the value of the loss function corresponding to the smaller step size is higher relative to $\Delta t = 1.0$, while the other methods do not, which also indicates that the model trained by the Euler method is more overfitted.

Table VI: The Wilcoxon signed-rank test results between loss values under different step sizes.

Training-Testing Solver Combinations	Time Step Size Pairs		
	(0.5, 1.0)	**(0.1, 1.0)**	**(0.1, 0.5)**
Euler-Euler	$L_{0.5} > L_{1.0}$	$L_{0.1} > L_{1.0}$	Not Significant
ImE-ImE	Not Significant	Not Significant	Not Significant
RK4-RK4	Not Significant	Not Significant	Not Significant

V. Conclusion

Our systematic research shows that, the choice of training solver is significantly more important for Noise-NCA model. Model trained using lower-order methods, such as the Euler method, are more overfitted, which affects the quality of the resulting textures. Unfortunately, a lot of previous works have used the Euler method to train similar models. We recommend using higher-order methods when training the model, such as the improved Euler method or the fourth-order Runge-Kuta method, to improve the generalization ability of the model. On the other hand, if the overfitting of the model is not severe, the choice of test solver does not have a significant impact on performance.

[1]Related testing can be found in the .ipynb file, which indicates that most of the differences between the paired samples do not follow the Gaussian distribution.

ACKNOWLEDGEMENTS

The author thanks Yitao Xu for his careful reading and helpful suggestions.

ETHICS COMPONENT

We have identified one ethical risk, particularly concerning the users as the stakeholders.

Welfare: The Noise-NCA model is designed to make static images move based on a certain motion. In theory, users can download the model, provide their own images and motions, and generate videos of any length. Users could potentially create harmful videos, such as generating videos using someone's face without their consent, although the model may not always produce ideal results, especially for complex images and motions. Additionally, because the model is open source, users are not subject to restrictions that could prevent unethical use.

Because the model is an improved version based on a Cellular Automaton (CA) model, where all cells of the entire image learn the same motion globally for updates, it is unlikely to generate highly realistic videos, particularly from portrait images. This reduces the potential for serious consequences.

We tested the model using a classic portrait photo as below and found that the generated video barely retains any facial information(Some representative frames are extracted and displayed). Therefore, we believe the harm caused by the model in this regard is relatively low. Since the model is

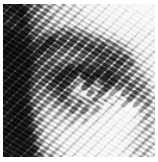

Figure 3: Example of the image.

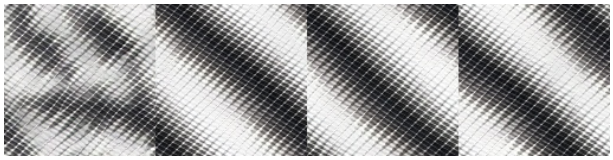

Figure 4: frames of the video.

fully open-source, the actions of users cannot be controlled by the creators of the model. If the model is not public, it is difficult to verify the repeatability of the model, and the user's AUTONOMY is also damaged.

Fairness: The model is publicly available to everyone online and does not produce different results based on the user's background or demographic. Therefore, it treats all users equally without discrimination.

Autonomy: Relevant papers and documentation provide explanations about the working principles and limitations of the Noise-NCA model. Users have the autonomy to choose images and motions for training and generating videos. Additionally, the texture database at https://www.robots.ox.ac.uk/ vgg/data/dtd provides texture images that can be used for training.

Privacy: The model does not require users to submit any data. The user can download and then train the model locally with images and motions provided by the user themselves , ensuring that user privacy is maintained.

Sustainability: In terms of sustainability, aside from the human labor involved in developing the model, the use of test images may require the labor of others who have prepared the images for testing. However, the primary focus on model development and testing is from the model developers themselves.

REFERENCES

[1] J. Von Neumann, A. W. Burks *et al.*, "Theory of self-reproducing automata," 1966.

[2] W. Gilpin, "Cellular automata as convolutional neural networks," *Physical Review E*, vol. 100, no. 3, p. 032402, 2019.

[3] A. Mordvintsev and E. Niklasson, "μ nca: Texture generation with ultra-compact neural cellular automata," *arXiv preprint arXiv:2111.13545*, 2021.

[4] E. Pajouheshgar, Y. Xu, and S. Süsstrunk, "Noisenca: Noisy seed improves spatio-temporal continuity of neural cellular automata," *arXiv preprint arXiv:2404.06279*, 2024.

[5] A. M. Turing, "The chemical basis of morphogenesis," *Bulletin of mathematical biology*, vol. 52, pp. 153–197, 1990.

[6] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, "Self-organising textures," *Distill*, vol. 6, no. 2, pp. e00 027–003, 2021.

[7] E. Pajouheshgar, Y. Xu, T. Zhang, and S. Süsstrunk, "Dynca: Real-time dynamic texture synthesis using neural cellular automata," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 20 742–20 751.

[8] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, "Growing neural cellular automata," *Distill*, vol. 5, no. 2, p. e23, 2020.