INVERTIBLE NEURAL CELLULAR AUTOMATA

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce *invertible Neural Cellular Automata* (iNCA), a novel recursive, self-organizing and space invariant generative neural architecture, which is able to learn invertible mappings with *Cellular Automata*. Following an bi-directional additive coupling strategy and modeling an additional latent target distribution, allows iNCA to learn ambiguous inverse mappings and the estimation of posterior probabilities. Compared to previous invertible neural network architectures, iNCAs show a significant improvement in inverse mapping capabilities for data organized in discrete grid structures. We demonstrate the general theoretical capabilities of iNCA empirically on a series of image generation and classification tasks.

1 INTRODUCTION

For many practically relevant applications, especially in natural sciences, a standard task is to determine hidden system parameters x of some parameter-space $\mathcal{X}$ from measurable observations y (from an observation-space $\mathcal{Y}$). However, from a learning perspective, such problems can turn out to be quite hard: while it is often times rather easy to learn well-defined *forward*-mappings $\mathcal{X} \rightarrow \mathcal{Y}$, e.g. following the *Universal Approximation Theorem for Neural Networks* [Hornik et al., 1989] and learn the function $y = f(x)$ from data, the inverse *backward* direction $\mathcal{Y} \rightarrow \mathcal{X}$ might suffer from strong ambiguities [Ardizzone et al., 2019]. In order to model all possible inverse solutions x conditioned by an observed $y \in \mathcal{Y}$, it is often necessary to estimate the complete posterior $p(x|y)$. Figure 1) illustrates the fundamental problem with a simple MNIST example: while generating images y from a 2D parameter-space $\mathcal{X}$ encoding digit label and rotation is a rather easy learning task which can be solved perfectly with generative Neural Networks (NN) (see results in table 2), learning the inverse, i.e. estimating the posterior $p(x|y)$ of all possible input parameters that could map to a given image, cannot be solved with a simple NN based classifier.

We follow a new approach to model such hard inverse learning problems by introducing *invertible Neural Cellular Automaton* (iNCA), a novel neural architecture which allows to learn ambiguous bidirectional mappings between data manifolds structured on discrete grids (like the image example in fig. 1). Our approach builds on the rich theoretical basis of learnable Cellular Automata (NCA) [Wolfram & Gad-el Hak, 2003; Mordvintsev et al., 2020; Spitznagel & Keuper, 2026]. By introducing invertible channel-wise additive coupling transformations and an additional latent target distribution, we show that iNCA are able to learn to estimate a population posterior distribution of complex inverse mappings for practical problems in the image domain.

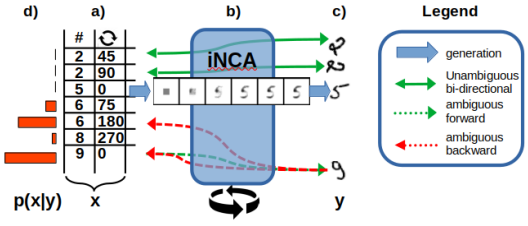

Figure 1: **Simplified sketch of an iNCA** application: generating images c) from a parameter-space encoding labels and rotations a) via the recursive iNCA calls b). The generation process is depicted for the example of a "5" without rotation. Our iNCA are not only able to invert unambiguous generations (solid green lines), but also to **sample likely inverse parameter estimates from ambiguous mappings** (red lines) and to estimate the posterior distribution d).

Our empirical results show that iNCA are able to robustly learn inverse mappings and estimate the posterior for such problems. Direct comparison with prior approaches like i-RevNet [Jacobsen et al., 2018], a RealNVP-based INN [Dinh et al., 2017] and Glow-based INN [Kingma & Dhariwal, 2018]

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

on MNIST [LeCun, 1998], Edge2Icon [Otte et al., 2021] and Edge2Handbag [Isola et al., 2017], show significant improvements.

Contributions:

- We are the first to approach inverse learning problems via discrete automata by introducing *invertible Neural Cellular Automata* (iNCA). Our novel method allows to model inverse mappings on discrete grid data, e.g. for applications in the highly relevant image domain.
- We show that iNCA are not only theoretically able to learn complex inverse mappings, but also produce practically usable posterior estimates from which we can draw meaningful samples.
- Our experiments show superior modeling capabilities of complex inverse mappings compared to several state-of-the-art methods even on high resolution image data.

1.1 Related Work

1.1.1 Cellular Automata.

Cellular Automata (CA) are discrete computational models which have been studied since the 1940s by *John von Neumann* [Von Neumann, 1966] and others. In its most basic form, a *Cellular Automaton* consists of a finite, regular grid $\mathcal{G}$ of discrete cells. Each cell i takes on a discrete state $s_{i,t}$ for a given discrete time-step t. Over the progress of time, cell states are recursively updated based on a set of rules $\mathcal{R}$ which use the local neighborhood $\mathcal{N}(s_{i,t})$ to compute the new cell state $s_{i,t+1} := \mathcal{R}[\mathcal{N}(s_{i,t})]$ for all cells synchronously [Schiff, 2011]. Figure 2 illustrates the computation process and rules of a simple CA.

CAs have been widely popularized in the 1970s by *Conway's "Game of Life"* [Gardner, 1970], which employed a 2D CA with binary states ($s_{(x,y),t} \in \{0, 1\}$) and update-rules using a 3×3 neighborhood. Following works not only showed wide application possibilities of CAs in biological [Coombes, 2009; Bouligand, 1986; Hatzikirou et al., 2012], chemical [Gerhardt & Schuster, 1989] and physical modeling [Wolfram, 1983; Załuska-Kotur et al., 2021], but also strong theoretical computational properties like *Turing Completeness* [Cook et al., 2004] for certain CA configurations, which established CAs as a universal computing model. This motivated *Stephen Wolfram* to postulate his famous *"A New Kind Of Science"* [Wolfram & Gad-el Hak, 2003], suggesting to replace intrinsically limited mathematical formulations with more powerful "simple recursive programs", as he formulated a new formal framework for a wide range of scientific applications based on *Cellular Automata*.

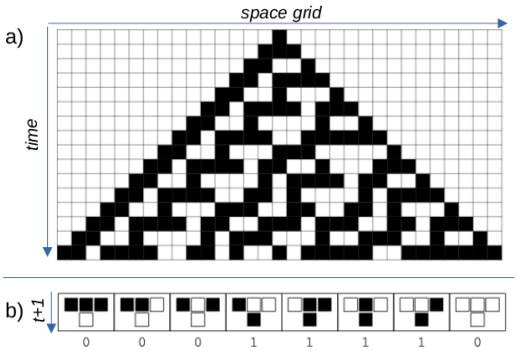

Figure 2: Visualization of a simple, 1D CA with binary states. a) shows the recursive update of grid states over time by applying the update rules $\mathcal{R}$ on the 3×1 cell neighborhoods shown in b).

1.1.2 Neural Cellular Automata

Traditional CA, as proposed by [Wolfram & Gad-el Hak, 2003], [Gardner, 1970], and [Von Neumann, 1966], are built with a fixed set of manually designed rules. For example, the update rule in fig. 2 is known as "Rule #30" (out of $2^{2^3} = 256$ possible rules) for a 1D, binary-state CA with a neighborhood size of 3 introduced in [Wolfram & Gad-el Hak, 2003]. While all possible rules for a given CA with predetermined discrete states $\mathcal{S}$ and neighborhood-size $|\mathcal{N}|$ can be constructed combinatorially, such rule-spaces unfortunately grow exponentially: even the very simple 2D binary CA behind the *Game of Life* already has $2^{2^{3*3}} = 2^{512}$ possible rules for its 3×3 neighborhood. Consequently, the idea to learn more complex rules from data instead of manually designing them has become increasingly appealing. [Mordvintsev et al., 2020] introduced the concept of *Neural*

Cellular Automata (NCA), which essentially replaces manually designed rules with artificial Neural Networks (NN) that are trained on data of a given problem. Equation 1 gives an abstract formalization of an NCA update function, where f_ϕ represents an arbitrary neural network with learnable parameters ϕ:

$$s_{i,t+1} := s_{i,t} + f_\phi[\mathcal{N}(s_{i,t})] \tag{1}$$

Since f_ϕ are space invariant with regard to $\mathcal{G}$ (during a time step, constant f_ϕ are applied to all positions i), [Mordvintsev et al., 2020] proposed an efficient implementation via "nested" *Convolutional Neural Networks* (CNNs) [LeCun & Bengio, 1998]. In its simplest form, the kernels (filters) of a single convolutional layer represent learnable rules for neighborhoods of the kernel size which are recursively applied. The theoretical equivalence of "nested" CNNs with CA has been theoretically proven by [Gilpin, 2019]. However, it should be noted that most NCA architectures are relaxing the original CA property of discrete cell states towards continuous vector state representations $s_{i,t} \in \mathbb{R}^n$.

While the original work of [Mordvintsev et al., 2020] used a simple 1-layer convolutional NCA to show impressive self-organizing and classification tasks on images, later works extended the use of CNNs to more complex network architectures: [Sandler et al., 2020] showed NCA based image segmentation with larger CNNs, while [Tesfaldet et al., 2022] introduced an *attention* based implementation. NCAs based on GANs [Otte et al., 2021], VAEs [Palm et al., 2022] or diffusion [Kalkhof et al., 2024] have also been utilized for generative tasks. [Grattarola et al., 2021] applied an NCA based approach on graph data.
We follow the modular formulation introduced by [Spitznagel & Keuper, 2026], as visualized in fig. 3:

$$s_{t+1} := s_t + \mathcal{U}_\phi[\mathcal{P}_{\mathcal{N},\phi} * s_t] \tag{2}$$

where the learnable rules are split into a **learnable perception** $\mathcal{P}_{\mathcal{N},\phi}$ of neighborhoods $\mathcal{N}$ matching the kernel size, followed by a **learnable update layer** $\mathcal{U}_\phi$ implemented as 1×1 convolutions.

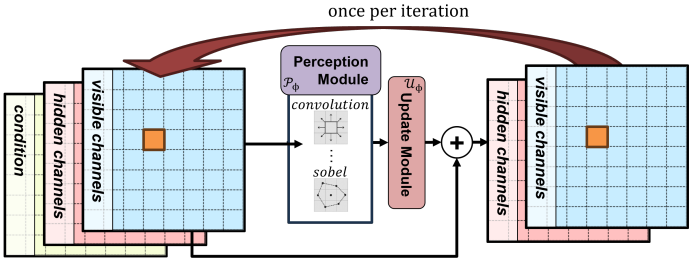

Figure 3: Sketch of a basic CNN implementation of a 2D NCA with a 3D state space. The perception module processes the cell state using one of several configurable perception options. The update module processes the perceived features to compute state updates. Stochastic updates and alive masking are applied before updating the visible and hidden channels to form the new cell state for the next iteration. The sample pool is used after one batch of images is processed to retain and reuse evolved states for subsequent training iterations.

1.1.3 Invertible Neural Networks

Following [Ardizzone et al., 2019], we define the necessary properties of an Invertible Neural Network (INN) as follows:

(i) the mapping from inputs to outputs is bijective, i.e. its inverse exists
(ii) both forward and inverse mapping are efficiently computable
(iii) the backward mapping allows explicit computation of posterior probabilities

A standard approach, which would allow such mappings and to sample likely x for a given observation y, are *Bayesian Neural Networks* (BNNs) [FV, 1996]. Unfortunately, this approach is unsuitable for many practical applications, due to the very high compute costs of BNNs [Gal & Ghahramani, 2016].
To overcome these limitations, [Dinh et al., 2017] introduced *Real-valued non-Volume Preserving*

transformations (NVP) which are able to learn *backward* mappings via simple *invertible* transformations. While this basic approach is often considered to be one of the first examples of an *Invertible Neural Network* (iNN), it still lacked the power to handle ambiguous mappings. The latter ability has then been introduced by [Ardizzone et al., 2019], who extended the NVP approach by adding an additional latent space: $\mathcal{X} \leftrightarrow [\mathcal{Y}; \mathcal{Z}]$ and enforcing a Gaussian distribution in $\mathcal{Z}$ via an additional training loss. This construct allows for approximating the full posterior of $p(x|y)$.
Another prominent approach towards learnable invertible mappings are *Normalizing Flows* (NF) [Rezende & Mohamed, 2015; Papamakarios et al., 2021]. NFs parameterize mappings between distributions as a series of concatenations of invertible transformations g_i:

$$\mathcal{X} \overset{g_0}{\longleftrightarrow} h_0 \overset{g_1}{\longleftrightarrow} h_1 \overset{g_2}{\longleftrightarrow} h_2 \dots \overset{g_k}{\longleftrightarrow} \mathcal{Y} \tag{3}$$

While the choice of a suitable invertible g heavily depends on the application, [Kingma & Dhariwal, 2018] introduced a generalized flow variant for grid-like data, named *Glow*, which utilizes invertible 1×1 convolutions. *Glow* demonstrated notable results on image modeling benchmarks.

2 Invertible Neural Cellular Automata

In the following, we introduce *Invertible Neural Cellular Automata* (iNCA), a novel architecture which extends NCA by invertible mapping capabilities. Our approach is fundamentally motivated by the finding in [Wolfram & Gad-el Hak, 2003], who showed the existence of so-called *"reversible"* update rules $\mathcal{R}^{-1}$ in standard CA which cyclically return to their initial state:

$$s_t \xrightarrow{\mathcal{R}^{-1}} s_{t+1} \xrightarrow{\mathcal{R}^{-1}} s_{t+2} \xrightarrow{\mathcal{R}^{-1}} \dots \xrightarrow{\mathcal{R}^{-1}} s_{t+m} = s_t \tag{4}$$

This also allows to propagate backwards through the state-time by recursive application of *"reversible"* update rules $s_{t-1} = (\mathcal{R}^{-1}[s_t])^{m-1}$. While [Wolfram & Gad-el Hak, 2003] showed the existence of such *"reversible"* rules, they also found that they make up only a very small portion of all possible rules. Using learnable NCA, we aim to find suitable invertible rules from training data.
The iterative mapping from some input-output relation over a series of time-steps t implemented in NCA (as shown in eq. 1 and fig. 3) already shows significant similarities to the formulation of *Glow* NFs (see eq. 3): the main difference is that *Glow* NFs use invertible 1×1 convolutions instead of general CNN architectures. While CNNs can represent cellular automaton rules [Gilpin, 2019], a learned update is not necessarily invertible. We therefore use additive coupling [Dinh et al., 2017; Jacobsen et al., 2018] to ensure that every CA update is invertible by construction, while preserving local, space-invariant interactions and shared weights across rollout steps. To additionally represent ambiguous inverse mappings, we adopt the latent space approach from [Ardizzone et al., 2019] (see section 5) to estimate the full posterior $p(x|y)$.

2.1 The coupling CA step

We split the cell state $s \in \mathbb{R}^n$ into two channel groups, $s = [s_1, s_2]$, and update them sequentially, with each residual computed from the group it does not modify (fig. 4):

$$s_1' = s_1 + F\big(\mathcal{N}(s_2), c\big), \tag{5}$$

$$s_2' = s_2 + G\big(\mathcal{N}(s_1'), c\big), \tag{6}$$

where $\mathcal{N}$ denotes a local neighborhood and c is an optional condition that must be held fixed during inversion. We implement this using the modular perception and update structure of NCA-torch [Spitznagel & Keuper, 2026], with $F = \mathcal{U}_{\phi_F} \circ \mathcal{P}_{\theta_F}$ and $G = \mathcal{U}_{\phi_G} \circ \mathcal{P}_{\theta_G}$. The perception $\mathcal{P}$ aggregates the local neighborhood into a feature vector at each cell. The update network $\mathcal{U}$ maps these features into an additive state update. The inverse evaluates the same residuals in reverse order and subtracts them:

$$s_2 = s_2' - G\big(\mathcal{N}(s_1'), c\big), \qquad s_1 = s_1' - F\big(\mathcal{N}(s_2), c\big). \tag{7}$$

Neither F nor G must be invertible, because the input required to recompute each residual remains available when that operation is undone (see section A.2 for the proof). We alternate the update order of the channel groups between steps to balance information exchange across the rollout. Reversing a T-step rollout applies the inverse steps in reverse temporal order with the same swap schedule. The

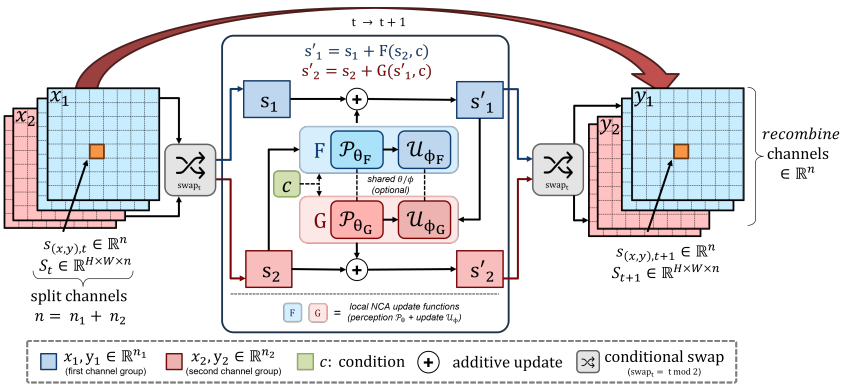

Figure 4: **One iNCA coupling step.** The cell state is divided into channel groups s_1 and s_2. The residual F updates s_1 from s_2, after which G updates s_2 from s'_1. Each residual combines a local perception operator $\mathcal{P}_\theta$ with an update network $\mathcal{U}_\phi$. The time-dependent swap exchanges the channel roles on alternating steps, and the same parameters are reused throughout the rollout. The optional condition c is unused in the reported experiments.

resulting bijection is exact over real arithmetic and approximate only up to floating-point round-off in its implementation. Reusing the same F and G separates the size of the learned rule: a T-step rollout has the parameter count of a single update, with computational cost linear in T. Repeated local updates expand the effective receptive field. Rollout length controls how far information can propagate without adding independently parameterized layers, while the neighborhood operator controls how far it can propagate per step.

Inversion from observations: At inference, the observed image y specifies only part of the complete CA state. We fill its unobserved channels with fixed values before inversion. The resulting state generally differs from a generated endpoint, so exact full-state invertibility alone does not guarantee recovery of the desired seed encoding x. We therefore supervise both image generation and seed recovery from completed observations using the same model parameters. The bidirectional objective is specified in section A.3.1.

2.2 Efficient iNCA Training

Several optimizations can be applied to improve the training efficiency of iNCA. As in other NCAs, computation and memory requirements increase with grid size and rollout length, while communication over greater spatial distances requires more iterations for a fixed local neighborhood [Spitznagel & Keuper, 2026]. A separate analysis of inference runtime and its dependence on rollout length of iNCA is provided in section A.3.2.

Memory-efficient backpropagation: Standard backpropagation retains intermediate activations across the rollout. iNCA can instead reconstruct preceding states through its inverse dynamics, following reversible residual networks [Gomez et al., 2017]. Table 1 compares this approach with ordinary autograd and gradient checkpointing. Reversible backpropagation uses the least memory among the tested configurations, at the cost of a longer training step. Implementation and evaluation details are given in section A.3.3.

Table 1: MNIST training-step memory and runtime using the same iNCA weights and inputs (RTX 4090, batch size 16, 36×36 grid, 41 updates per direction). Memory is peak allocated GPU tensor memory. Runtime is the median over five repetitions. Full protocol in section A.3.3.

Backward method	Peak memory (MiB)	Step time (ms)
Ordinary autograd	6980.4	202.2
Gradient checkpointing	295.3	220.9
Reversible backpropagation	135.4	240.4

Training on latent grids : Latent-space training, previously proposed for NCAs [Menta et al., 2024], also applies to iNCA. The same invertible update rule can operate on a compressed spatial representation, reducing the number of cells updated per step and the number of local interactions needed to span the grid. We demonstrate this on Edge2Icon using a pretrained, frozen VQ-VAE [Van den Oord et al., 2017]: iNCA maps between encoded edges and icons on a 35×35 grid, and the decoder maps the results to 280×280 images (fig. 10). The CA dynamics remain invertible

in this representation, while image reconstruction depends on the autoencoder. Qualitative results and the scaling analysis are provided in section A.4.1.

3 BIDIRECTIONAL MNIST AND ARCHITECTURAL ABLATIONS

We evaluate iNCA on a bidirectional MNIST task [LeCun, 1998], where the target $y \in \{0, 1, \ldots, 255\}^{28\times28}$ is a MNIST digit image padded to a 36×36 grid, while the seed $x \in \{e_1, e_2, \ldots, e_{10}\}^{9\times9}$ with $e_i \in \mathbb{R}^{10}$ is a centered 9×9 patch containing a one-hot class code. The forward task is to generate a digit from the label $x \to y$ and then invert the mapping to recover the seed $y \to x$.

Protocol. All configurations in table 2 use the same data, supervision, losses, and trainer for 100k optimization steps. They contain 32 state channels and are trained with a batch size of 16 and a learning rate of 5×10^{-4}. We train iNCA over a range of 32–41 updates. For each training batch, the number of CA updates is chosen uniformly from this range and used in both directions. We use the upper endpoint, 41 updates, as a common evaluation budget for all iNCA variants and set the feedforward baselines to 41 blocks to match this depth. We report the mean and standard deviation over four independent runs per configuration.

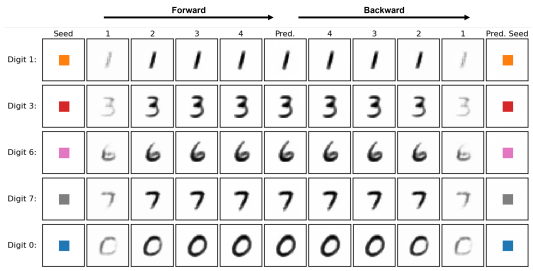

Figure 5: Bidirectional MNIST trajectories. Each row shows the initial class-encoding seed, intermediate states during generation, and the inverse trajectory from the digit to the recovered seed. Additional examples are provided in section A.4.

Baseline. We compare against i-RevNet [Jacobsen et al., 2018], which uses independent parameters at each depth. Its blocks follow the authors' released pre-activation bottleneck with additive coupling and swapped channel groups. We also compare against a RealNVP-based INN [Dinh et al., 2017] and a Glow-based INN [Kingma & Dhariwal, 2018] with a parameter budget close to the 5×5 iNCA. Implementation details are given in sections A.6 and A.7.

All configurations attain $100.00\,\%$ forward accuracy according to supervised classifier evaluation (see section A.5). Forward generation does not distinguish the models in this experiment, and the following analysis focuses on inverse recovery. The inverse task recovers the class-encoding seed from an unseen digit image using the same weights as the forward model. We evaluate 200 test images per class and report the class read from the recovered seed. Figure 5 shows representative trajectories in both directions.

Table 2: MNIST architecture and parameter-sharing comparison under the same training protocol. Values are mean $\pm$ sample standard deviation over four runs. Baseline width w denotes the number of hidden channels per coupling subnet.

Model	Params.	Fwd. Acc. ↑ (%)	Bwd. Acc. ↑ (%)
iNCA — neighbourhood variants			
iNCA (sobel)	21k	100.00	77.3 ± 8.6
iNCA (3×3)	74k	100.00	82.0 ± 7.7
iNCA (5×5)	140k	100.00	95.1 ± 2.5
iNCA ($3 \times 3 + 5 \times 5$)	210k	100.00	92.7 ± 5.1
iNCA — 5×5 sharing between F and G			
iNCA (5×5), $\theta_F = \theta_G$	88k	100.00	$\mathbf{95.5 \pm 1.4}$
iNCA (5×5), $\phi_F = \phi_G$	121k	100.00	91.0 ± 5.4
iNCA (5×5), $F = G$	70k	100.00	86.2 ± 9.6
i-RevNet (w=8)	119k	100.00	76.1 ± 8.6
i-RevNet (w=128)	7.57M	100.00	88.2 ± 6.3
RealNVP	135k	100.00	87.2 ± 4.7
Glow	133k	100.00	92.6 ± 2.0

Table 2 compares neighborhood choices, sharing between coupling residuals, and untied invertible baselines. The 5×5 iNCA with separate F and G has a higher mean inverse accuracy than both i-RevNet widths and RealNVP. This demonstrates competitive recovery with a compact recurrent rule. Within iNCA, widening the learned neighborhood improves mean recovery, while adding a second perception branch provides no further gain.

The sharing variants further reduce the 5×5 model's parameter count. Sharing only the perception operators preserves mean inverse accuracy and provides the highest mean in the table. Sharing the update network or the complete residual lowers the mean, with substantial variation across runs.

These variants change parameter sharing between F and G. Latency and the effect of rollout length are evaluated separately in section A.3.2.

4 Structured inversion on Edge2Icon

MNIST requires the inverse to recover a categorical seed. Next, we test a structured image-to-image mapping using Edge2Icon [Otte et al., 2021], where $x \in \{0, 1, \ldots, 255\}^{4\times80\times80}$ is a binary edge map encoded in RGBA and $y \in \{0, 1, \ldots, 255\}^{4\times80\times80}$ is the corresponding rendered RGBA emoji. We train on 79 emoji classes and evaluate generalization on 34 held-out classes. We distinguish three tasks. Forward LPIPS [Zhang et al., 2018] measures icon generation. Real-inverse IoU measures edge recovery when a ground-truth icon is completed with the fixed hidden-channel fill and inverted. Projected-round-trip IoU first generates an endpoint, retains only its visible RGBA channels, replaces the hidden channels with the same fill, and then inverts it. The last task measures how much seed information survives the observation bottleneck.

All models recover held-out edge maps from real icons with similar IoU, ranging from 0.9441 to 0.9728. After generated endpoints are reduced to their visible channels, however, expanding the iNCA neighborhood from 3×3 to 5×5 raises IoU from 0.7732 to 0.8686, while combining both neighborhoods yields 0.8749. Edge2Icon therefore supports a narrower conclusion: wider context helps preserve seed structure through the visible-channel bottleneck, not inversion from a ground-truth icon.

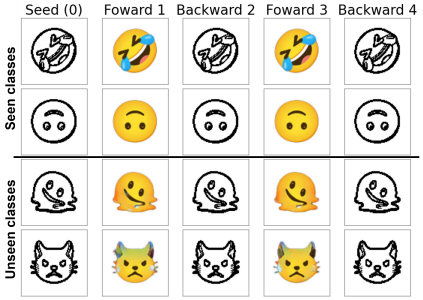

Figure 6: Bidirectional Edge2Icon example showing forward evolution and inverse reconstruction.

Table 7 shows that increasing i-RevNet width mainly improves fitting of the training classes, with modest gains on held-out classes. Increasing RealNVP width improves mean projected-round-trip IoU while real-inverse IoU remains similar. Both RealNVP widths trail the 5×5 iNCA in held-out forward LPIPS and projected-round-trip IoU. The narrow RealNVP provides this comparison at a similar parameter budget, although its mean real-inverse IoU is slightly higher. These gains in generation and recovery through the observation bottleneck therefore hold at a comparable parameter budget and persist against the wider RealNVP baseline.

Table 3: Edge2Icon results on 34 held-out emoji classes. Real inverse starts from the paired ground-truth icon. Projected round trip discards the hidden channels of a generated endpoint before inversion. Values are mean ± sample standard deviation over four runs. Baseline width w denotes the number of hidden channels per coupling subnet.

Model	Forward LPIPS ↓	Real-inverse IoU ↑	Projected-round-trip IoU ↑	Params.
iNCA (Sobel)	**0.1350 ± 0.0030**	0.9716 ± 0.0023	0.7928 ± 0.0110	21k
iNCA (3×3)	0.1585 ± 0.0105	0.9727 ± 0.0023	0.7732 ± 0.0218	74k
iNCA (5×5)	0.1506 ± 0.0081	0.9654 ± 0.0014	0.8686 ± 0.0039	140k
iNCA ($3 \times 3 + 5 \times 5$)	0.1485 ± 0.0118	0.9636 ± 0.0020	**0.8749 ± 0.0097**	210k
i-RevNet (w=8)	0.1640 ± 0.0096	0.9630 ± 0.0050	0.8177 ± 0.0149	119k
i-RevNet (w=128)	0.1489 ± 0.0051	0.9441 ± 0.0093	0.8399 ± 0.0108	7.57M
RealNVP	0.1706 ± 0.0067	**0.9728 ± 0.0014**	0.7854 ± 0.0194	135k
Glow	0.1980 ± 0.0129	0.9525 ± 0.0026	0.7611 ± 0.0176	134k

5 Learning Ambiguous Inverse Mappings with iNCA

Completing an observation with a fixed hidden state yields one deterministic inverse output and cannot represent multiple valid explanations. In the ambiguous MNIST experiment, following Ardizzone et al. [2019], we reserve d endpoint channels for a latent variable $z \in \mathbb{R}^d$. The forward rollout produces the visible target and a code $\hat{z}$. An MMD objective [Gretton et al., 2012] matches the aggregate distribution of these codes to $\mathcal{N}(0, I)$. Sampling z and evaluating the inverse for a fixed image can then produce different candidate seeds (fig. 7).

For this experiment, we fix the seed-side latent channels to zero so that the forward rollout must produce the endpoint code. Otherwise, the bijection could transport sampled noise directly to the

endpoint and satisfy MMD without learning a relationship between z and the image. We perturb the produced code before inversion to discourage its use solely as an identifier for a training example.

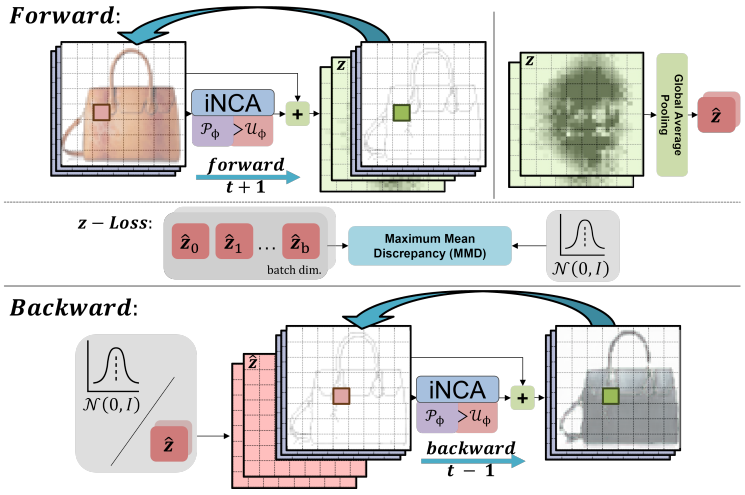

Figure 7: Latent-variable training and inference. The forward rollout produces a code $\hat{z}$ alongside the visible target, and an MMD objective matches the aggregate code distribution to $\mathcal{N}(0, I)$. At inference time, samples $z \sim \mathcal{N}(0, I)$ are combined with a fixed observation and passed through the inverse to generate candidate seeds.

We construct an ambiguous task from MNIST digits 6 and 9 at rotations of $0°$ and $180°$. Here, x encodes the digit–rotation pair and y denotes the observed image. The four joint classes form two visually equivalent pairs, $(6@0°, 9@180°)$ and $(6@180°, 9@0°)$. Each image is assigned one of its two valid readings uniformly during training. For a fixed observation, the desired conditional distribution therefore assigns approximately half of its mass to each member of the corresponding pair and no mass to the other pair. We test the total mass assigned to the valid pair, whether both members are recovered, and the balance between them. A $50/50$ split is meaningful only when the samples remain within the pair selected by the observation.

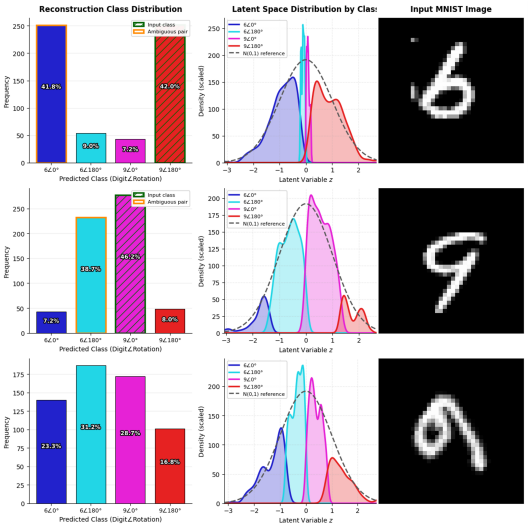

Figure 8: Latent-variable diagnostic for ambiguous 6/9 observations. For each fixed image, the recovered class is evaluated using samples $z \sim \mathcal{N}(0, 1)$. Comparing the outcomes across inputs reveals whether the admissible pair depends on the observation or whether the model has learned an image-independent prior. A full-width version is provided in fig. 12.

To estimate uncertainty for a fixed ambiguous observation y, we draw multiple samples $z \sim \mathcal{N}(0, I)$ and compute $x = f^{-1}(y, z)$. We classify each recovered seed and count how often each joint digit–rotation label occurs.

We use three complementary measurements. *Valid-pair mass* is the fraction of predictions belonging to either label compatible with the image. For an upright 6, these are $6@0°$ and $9@180°$. *Both-readings rate* is the fraction of test images for which the samples include both compatible labels at least once. The aggregate *split* measures how the valid mass is divided between these two labels across test images. Together, these measurements test whether predictions remain consistent with the image, whether both explanations are accessible, and whether one explanation dominates overall.

We evaluate four 177k-parameter 5×5 runs. Without a latent variable, $51.0\,\%$ of predictions belong to the valid pair, and the deterministic inverse cannot recover both readings of any image. With $d = 1$, $90.21\,\%$ of predictions belong to the valid pair and recover both readings for every test image.

Across all images and samples, 45.24 % select the original label and 44.97 % select its equivalent rotated alternative. These results show that latent sampling produces reasonable and stable solutions to ambiguous mappings. A per-image distributional evaluation is reported in section A.4.2.

5.1 Continuous appearance variation on Edge2Handbag

The ambiguous MNIST task tests a choice between discrete interpretations. We next study continuous appearance variation on Edge2Handbag [Isola et al., 2017], where an edge map constrains a handbag's structure but leaves its colour and texture unspecified. Here, x denotes the handbag image and y its edge map. We test whether z captures this appearance information and allows different handbags to be generated from the same edge.

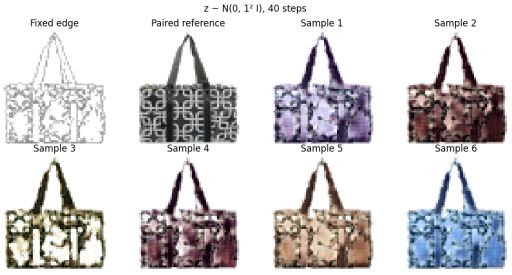

Figure 9: Held-out edges, paired handbags, reconstructions using encoded appearance, and four Gaussian samples per edge. Colour and appearance vary with z, while fine texture is not always reconstructed.

We use four 177k-parameter 5×5 iNCA on 64×64 images with an eight-dimensional Gaussian code and 40 rollout steps. The same model encodes a real handbag into an edge–latent pair and generates a handbag from an edge and code. Table 4 compares reconstruction using the paired handbag's code, a zero code, or a code from another handbag while keeping the edge fixed. Figure 9 shows appearance variation from Gaussian samples. Training combines paired reconstruction, adversarial supervision, latent matching, and code recovery through the visible image. Further details in section A.8.

The paired code gives lower reconstruction error than either control, supporting the use of z for appearance information absent from the edge map. This comparison tests code use rather than image quality relative to other generators. Additional diagnostics in table 9 show strong edge recovery from generated RGBA after hidden-channel removal, but substantially weaker recovery from real handbags.

Table 4: For each of 200 validation pairs, we hold the true edge and model fixed and vary only the reconstruction code. Values are mean ± sample standard deviation across four runs.

Appearance code	Reconstruction LPIPS ↓
Encoded from the paired handbag	**0.1813 ± 0.0044**
Zero code	0.2443 ± 0.0072
Encoded from another handbag	0.2785 ± 0.0037

6 Conclusion

We introduced iNCA, a novel recursive, self-organizing and space invariant generative neural architecture, which is able to learn invertible mappings on discrete grid data like images. Our method learns image generation and structured inversion through recursively applying learned local rules shared across space and time. Additive coupling makes the reverse process the explicit inverse of the forward dynamics, while bidirectional training adapts both directions to the task. Our empirical evaluations on the MNIST and Edge2Icon benchmarks demonstrate robust and competitive inverse recovery abilities. Reconstructing intermediate states during training also reduces activation storage, addressing a memory bottleneck of cellular models as grid size and rollout length increase.
Latent sampling extends this approach to ambiguous mappings, producing meaningful candidates in experiments with alternative digit interpretations and varying handbag appearances for a fixed observation. These experiments give the learned inverse two practical roles: recovering a structured explanation of an image and exploring alternatives when that explanation is not uniquely determined. iNCA brings these capabilities into a cellular model whose generation and inversion share the same learned dynamics.

Limitations. While the invertibility of iNCA is theoretically guaranteed (see proof section A.2), exact inverse computation has practical limits induced by numerical precision depending on the number of recursive iterations. Exact invertibility also holds for the iNCA cell states, not necessarily for a projection onto visible channels like described in the latent space NCA setting. Resource costs depend on rollout length, feature width, and execution settings.

AI USE STATEMENT

In this work, we used generative AI tools as a coding assistant and for language editing, to revise and shorten passages of the manuscript that we had drafted ourselves. Generative AI tools were not used to generate research ideas, to design the proposed architecture, to produce or interpret experimental results, or to retrieve or summarize related work. All code was reviewed and tested by the authors, and every reported number was produced by running that code. We have reviewed all AI-assisted text and take full responsibility for the final content of this work, including all text, claims and artifacts.

ETHICS STATEMENT

This work involves no human subjects, no personally identifiable information, and no user data. All datasets used are public and were employed within their intended scope. We release no new dataset. The proposed models are small and were trained on a single consumer GPU, so the computational footprint of this work is modest. We see no direct pathway to harmful application, although improved invertible performance is a general capability whose downstream uses we cannot fully anticipate. The authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

The appendix specifies the coupling architecture, baseline implementations, training objectives, and evaluation protocols. The full source code, including the configuration files reproducing every reported run, will be released upon publication as a plugin for the *NCATorch* framework of [Spitznagel & Keuper, 2026].

REFERENCES

Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W. Pellegrini, Ralf S. Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks, 2019. URL https://arxiv.org/abs/1808.04730.

Y Bouligand. Fibroblasts, morphogenesis and cellular automata. In *Disordered Systems and Biological Organization*, pp. 367–379. Springer, 1986.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL https://arxiv.org/abs/1604.06174.

Matthew Cook et al. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

Stephen Coombes. The geometry and pigmentation of seashells. *Nottingham: Department of Mathematical Sciences, University of Nottingham*, 2009.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

Jensen FV. *An introduction to Bayesian networks*. London, UK: UCL Press, 1996.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Martin Gardner. Mathematical games - the fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223(4):4, 1970.

Martin Gerhardt and Heike Schuster. A cellular automaton describing the formation of spatially ordered structures in chemical systems. *Physica D: Nonlinear Phenomena*, 36(3):209–221, 1989.

William Gilpin. Cellular automata as convolutional neural networks. *Physical Review E*, 100(3): 032402, 2019.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf.

Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Learning graph cellular automata. *Advances in Neural Information Processing Systems*, 34:20983–20994, 2021.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. URL http://jmlr.org/papers/v13/gretton12a.html.

Haralampos Hatzikirou, David Basanta, Matthias Simon, Karl Schaller, and Andreas Deutsch. 'go or grow': the key to the emergence of invasion in tumour progression? *Mathematical medicine and biology: a journal of the IMA*, 29(1):49–65, 2012.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-revnet: Deep invertible networks, 2018. URL https://arxiv.org/abs/1802.07088.

John Kalkhof, Arlene Kühn, Yannik Frisch, and Anirban Mukhopadhyay. Frequency-time diffusion with neural cellular automata. *arXiv preprint arXiv:2401.06291*, 2024.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *Advances in neural information processing systems*, 31, 2018.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1998.

Andrea Menta, Alberto Archetti, and Matteo Matteucci. Latent neural cellular automata for resource-efficient image restoration, 2024. URL https://arxiv.org/abs/2403.15525.

Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 5(2):e23, 2020.

Maximilian Otte, Quentin Delfosse, Johannes Czech, and Kristian Kersting. Generative adversarial neural cellular automata. *arXiv preprint arXiv:2108.04328*, 2021.

Rasmus Berg Palm, Miguel González Duque, Shyam Sudhakaran, and Sebastian Risi. Variational neural cellular automata. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=7fFO4cMBx_9.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pp. 1530–1538. PMLR, 2015.

Mark Sandler, Andrey Zhmoginov, Liangcheng Luo, Alexander Mordvintsev, Ettore Randazzo, et al. Image segmentation via cellular automata. *arXiv preprint arXiv:2008.04965*, 2020.

Joel L Schiff. *Cellular automata: a discrete view of the world*. John Wiley & Sons, 2011.

Martin Spitznagel and Janis Keuper. A new kind of network? review and reference implementation of neural cellular automata. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=NRwjj0ZLq0.

Mattie Tesfaldet, Derek Nowrouzezahrai, and Chris Pal. Attention-based neural cellular automata. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 8174–8186. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/361e5112d2eca09513bbd266e4b2d2be-Paper-Conference.pdf.

Aaron Van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/7a98af17e63a0ac09ce2e96d03992fbc-Paper.pdf.

John Von Neumann. Theory of self-reproducing automata. 1966.

Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3): 601, 1983.

Stephen Wolfram and M Gad-el Hak. A new kind of science. *Appl. Mech. Rev.*, 56(2):B18–B19, 2003.

Magdalena Załuska-Kotur, Hristina Popova, and Vesselin Tonchev. Step bunches, nanowires and other vicinal "creatures"—ehrlich–schwoebel effect by cellular automata. *Crystals*, 11(9):1135, 2021.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation, 2018. URL https://arxiv.org/abs/1711.11586.

A TECHNICAL APPENDICES AND SUPPLEMENTARY MATERIAL

Contents

A.1 Video Material 13
A.2 Invertibility of iNCA 13
A.2.1 Proof sketch 13
A.2.2 Full proof 14
A.3 iNCA Implementation Details 16
A.3.1 Training objective 16
A.3.2 Training efficiency 16
A.3.3 Reversible backpropagation 17
A.4 Additional Results 17
A.4.1 Latent-space scaling 17
A.4.2 Per-image distributions for ambiguous MNIST 17
A.4.3 Continuous style variation 18
A.4.4 Complete Edge2Icon results 18
A.4.5 Bidirectional stability 18
A.4.6 Robustness analysis 20
A.5 MNIST forward classification accuracy 20
A.6 The RealNVP baseline 21
A.7 The Glow baseline 21
A.8 Edge2Handbag training and evaluation 21

A.1 VIDEO MATERIAL

We provide an anonymous video showing empirical results. Full video: `https://youtu.be/lK34RGTwM7Q?si=o67cKYQv68I_NpzA`

Jump markers to specific topics in that same video:

- 0:00 Bidirectional MNIST
- 0:41 Edge2Icon: training emotes
- 1:03 Edge2Icon: unseen emotes
- 2:05 Ambiguous MNIST
- 2:41 Edge2Handbag: style variations
- 3:41 Latent-space Edge2Icon

A.2 INVERTIBILITY OF INCA

A.2.1 PROOF SKETCH

Following the coupling-layer constructions of Real NVP and invertible neural networks, iNCA updates one part of its state using the other, allowing each update to be recomputed and undone.

Let $S = (S_1, S_2)$ denote the complete grid state, partitioned into two channel groups. For fixed parameters and conditioning information c, one iNCA step is

$$S_1' = S_1 + F\big(\mathcal{N}(S_2), c\big), \tag{8}$$

$$S_2' = S_2 + G\big(\mathcal{N}(S_1'), c\big), \tag{9}$$

where $\mathcal{N}$ extracts local neighborhoods and F, G compute residual updates. This is the additive special case of affine coupling.

Given any output (S_1', S_2'), the input is uniquely recovered by undoing the updates in reverse order:

$$S_2 = S_2' - G\big(\mathcal{N}(S_1'), c\big), \tag{10}$$

$$S_1 = S_1' - F\big(\mathcal{N}(S_2), c\big). \tag{11}$$

The first expression recovers S_2 because S_1' is already available; the second then recovers S_1. Substitution verifies both $H^{-1} \circ H = \mathrm{id}$ and $H \circ H^{-1} = \mathrm{id}$ for the single-step map H. Hence each step is bijective.

Neither $\mathcal{N}$ nor F or G needs to be invertible: they are only reevaluated on the same inputs used in the forward computation. Spatial interactions do not alter this argument, since the entire channel group supplying the neighborhoods remains unchanged during each half-step.

A finite rollout is a composition of bijective steps,

$$\Phi_T = H_{T-1} \circ \cdots \circ H_0, \tag{12}$$

and is therefore bijective, with

$$\Phi_T^{-1} = H_0^{-1} \circ \cdots \circ H_{T-1}^{-1}. \tag{13}$$

Thus, inversion applies the inverse steps in reverse temporal order, respecting the original channel-swap schedule. Parameter sharing across steps does not affect invertibility.

Consequently, iNCA is invertible by construction on its complete state space, independently of training, provided the same conditions and update rules are used during inversion. This guarantee is exact in real arithmetic and subject to floating-point rounding errors in practice. It does not imply unique recovery from visible channels alone: discarding hidden channels removes information, so seed recovery from partial observations is a separate learning problem. □

A.2.2 Full proof

Proposition (invertibility of iNCA). Consider a fixed finite grid with $d_1 + d_2$ state coordinates, partitioned into two channel groups. Its complete state space is

$$\mathcal{S} = \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}. \tag{14}$$

Fix the model parameters, neighborhood and boundary conventions, and any conditioning information c. Let

$$f_c : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}, \qquad g_c : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \tag{15}$$

be the deterministic residual maps

$$f_c(S_2) = F\big(\mathcal{N}(S_2), c\big), \qquad g_c(S_1) = G\big(\mathcal{N}(S_1), c\big). \tag{16}$$

Assume these maps are defined on their entire respective domains. Then the iNCA coupling step

$$H_c(S_1, S_2) = \big(S_1 + f_c(S_2),\ S_2 + g_c\big(S_1 + f_c(S_2)\big)\big) \tag{17}$$

is a bijection on $\mathcal{S}$. Every finite composition of such steps, including a prescribed alternation of the channel-update order, is also bijective.

Proof. The argument specializes the affine coupling construction used in Real NVP and invertible neural networks to additive coupling.

1. Each additive half-step is bijective. Define

$$A_c(S_1, S_2) = \big(S_1 + f_c(S_2), S_2\big), \qquad B_c(S_1, S_2) = \big(S_1, S_2 + g_c(S_1)\big). \tag{18}$$

The first half-step leaves S_2 unchanged, so its inverse is

$$A_c^{-1}(U_1, U_2) = \big(U_1 - f_c(U_2), U_2\big). \tag{19}$$

Indeed, for every $(S_1, S_2) \in \mathcal{S}$,

$$A_c^{-1}\big(A_c(S_1, S_2)\big) = \big(S_1 + f_c(S_2) - f_c(S_2), S_2\big) = (S_1, S_2), \tag{20}$$

and, for every $(U_1, U_2) \in \mathcal{S}$,

$$A_c\big(A_c^{-1}(U_1, U_2)\big) = \big(U_1 - f_c(U_2) + f_c(U_2), U_2\big) = (U_1, U_2). \tag{21}$$

Likewise,

$$B_c^{-1}(U_1, U_2) = \big(U_1, U_2 - g_c(U_1)\big), \tag{22}$$

and direct substitution gives

$$B_c^{-1} \circ B_c = B_c \circ B_c^{-1} = \mathrm{id}_{\mathcal{S}}. \tag{23}$$

Thus both half-steps are bijections.

2. A complete coupling step has an explicit two-sided inverse. The iNCA step is

$$H_c = B_c \circ A_c. \tag{24}$$

Consequently,

$$H_c^{-1} = A_c^{-1} \circ B_c^{-1}. \tag{25}$$

Explicitly, given an output (U_1, U_2), define

$$V_2 = U_2 - g_c(U_1), \tag{26}$$
$$V_1 = U_1 - f_c(V_2). \tag{27}$$

Then

$$H_c^{-1}(U_1, U_2) = \big(U_1 - f_c\big(U_2 - g_c(U_1)\big),\ U_2 - g_c(U_1)\big)\,. \tag{28}$$

For completeness, if $(U_1, U_2) = H_c(S_1, S_2)$, then

$$V_2 = S_2 + g_c(U_1) - g_c(U_1) = S_2, \tag{29}$$
$$V_1 = S_1 + f_c(S_2) - f_c(V_2) = S_1. \tag{30}$$

Conversely, starting from any (U_1, U_2) and constructing (V_1, V_2) as above, the forward step gives

$$V_1 + f_c(V_2) = U_1, \tag{31}$$
$$V_2 + g_c\big(V_1 + f_c(V_2)\big) = U_2 - g_c(U_1) + g_c(U_1) = U_2. \tag{32}$$

Hence

$$H_c^{-1} \circ H_c = H_c \circ H_c^{-1} = \mathrm{id}_{\mathcal{S}}, \tag{33}$$

which proves global bijectivity.

3. Local spatial interactions preserve the argument. Here S_1 and S_2 represent channel groups over the entire grid. During A_c, every coordinate of S_2 remains unchanged; therefore all neighborhoods used to evaluate $f_c(S_2)$ remain available. During B_c, the updated first group remains unchanged, so all neighborhoods needed to evaluate g_c remain available. Overlapping neighborhoods and shared convolutional weights therefore do not affect the inverse identities.

In particular, neither the neighborhood operator nor the neural networks F and G need to be injective, differentiable, or invertible. The inverse only reevaluates them; it never inverts them.

4. Alternating the update order preserves bijectivity. Updating the second group first gives the step

$$\widetilde{H}_c = A_c \circ B_c, \tag{34}$$

which is bijective with

$$\widetilde{H}_c^{-1} = B_c^{-1} \circ A_c^{-1}. \tag{35}$$

If channel exchanges are implemented explicitly as coordinate permutations, these are also bijections, with inverse given by the inverse permutation. Thus any prescribed composition of coupling half-steps and channel permutations is bijective.

5. Every finite rollout is bijective. Let $H_t : \mathcal{S} \to \mathcal{S}$ denote the complete bijective step used at time t, including its prescribed update order and any channel permutations. For $T \geq 1$, define

$$\Phi_T = H_{T-1} \circ \cdots \circ H_0. \tag{36}$$

Set

$$\Psi_T = H_0^{-1} \circ \cdots \circ H_{T-1}^{-1}. \tag{37}$$

By associativity of composition and successive cancellation,

$$\Psi_T \circ \Phi_T = H_0^{-1} \circ \cdots \circ H_{T-1}^{-1} \circ H_{T-1} \circ \cdots \circ H_0 = \mathrm{id}_{\mathcal{S}}, \tag{38}$$

$$\Phi_T \circ \Psi_T = H_{T-1} \circ \cdots \circ H_0 \circ H_0^{-1} \circ \cdots \circ H_{T-1}^{-1} = \mathrm{id}_{\mathcal{S}}. \tag{39}$$

Therefore Φ_T is bijective and $\Phi_T^{-1} = \Psi_T$. The inverse applies the inverse steps in reverse temporal order. The argument holds whether the parameters are shared across time or differ between steps. Thus invertibility is an architectural property and does not require a training loss. □

Scope of the guarantee. The proposition concerns the complete state and a fixed rollout length, with the same parameters, conditions, boundary conventions, and update schedule used in both directions. It establishes exact invertibility over real arithmetic, rather than a bound on numerical reconstruction error in floating-point arithmetic.

It also does not establish invertibility from visible channels alone. If π discards hidden coordinates, the observed mapping is $\pi \circ \Phi_T$. On the full state space this mapping is not injective: choose distinct endpoint states $U \neq V$ with $\pi(U) = \pi(V)$. Since Φ_T is bijective, their preimages are distinct, but

$$(\pi \circ \Phi_T)\big(\Phi_T^{-1}(U)\big) = \pi(U) = \pi(V) = (\pi \circ \Phi_T)\big(\Phi_T^{-1}(V)\big). \tag{40}$$

Recovery from visible observations on a restricted set of task-specific seeds therefore requires additional assumptions or learning objectives; it does not follow from architectural invertibility alone.

A.3 iNCA Implementation Details

A.3.1 Training objective

For MNIST and Edge2Icon, we sample one rollout length for each paired seed and target image (s_0, y) and train the same iNCA in both directions. The forward rollout $\tilde{s}_T = \Phi(s_0)$ is supervised against the target in its visible channels. The backward rollout starts from the ground-truth image with fixed values in all unobserved channels and is supervised against the task-relevant channels of the original seed. Denoting these terms by $\mathcal{L}_{\mathrm{fwd}}$ and $\mathcal{L}_{\mathrm{bwd}}$, the objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{fwd}} + \lambda_{\mathrm{bwd}}\mathcal{L}_{\mathrm{bwd}} + \lambda_{\mathrm{range}}\mathcal{L}_{\mathrm{range}} + \lambda_z \,\mathrm{MMD}^2(\{\hat{z}_b\}_{b=1}^B, \{z_b\}_{b=1}^B), \tag{41}$$

Both reconstruction terms use L_1 distance. Because the seed is sparse, $\mathcal{L}_{\mathrm{bwd}}$ averages errors over its nonzero and zero entries separately and adds the two averages. The range penalty keeps visible endpoint values near $[0, 1]$ and hidden endpoint values near $[-1, 1]$.

In ambiguous MNIST, the forward rollout also produces a code $\hat{z}$ in designated endpoint channels. MMD matches the batch of spatially averaged codes to Gaussian samples using a sum of three RBF kernels with bandwidths 0.5, 1, and 2 times the pooled median pairwise distance. During training, the backward rollout receives the produced code with Gaussian perturbation of standard deviation 0.25. At inference it receives a fresh draw from $\mathcal{N}(0, I)$. This base objective has no cycle-consistency term. The handbag extension additionally supervises projected image and latent cycles (section A.8); neither objective requires a loss to enforce architectural invertibility.

A.3.2 Training efficiency

An iNCA training step evaluates several rollouts and can therefore be costly relative to a feed-forward model with the same parameter count. Computation scales with rollout length rather than the number of weights. Section A.3.3 separately evaluates reducing activation storage through inverse state reconstruction. The measurements below concern inference cost.

Table 5 shows similar mean recovery at the shorter rollout length, with reduced latency. In this implementation, iNCA has lower single-image latency than the reference RealNVP architectures. Sharing perception reduces weight storage while leaving the internal feature dimensions unchanged. These measurements distinguish the size of the learned rule from the execution cost of applying it.

Table 5: MNIST inverse accuracy and inference cost within the trained rollout range. Accuracy is mean ± sample standard deviation over four runs.

Model	Steps/blocks	Inverse accuracy (%)	Batch 1 (ms)
iNCA (5×5)	32	95.08 ± 1.70	4.145
iNCA (5×5)	41	95.08 ± 2.45	5.306
iNCA (5×5), $\theta_F = \theta_G$	32	94.97 ± 1.80	4.128
iNCA (5×5), $\theta_F = \theta_G$	41	95.44 ± 1.42	5.278
RealNVP (w=5)	41	87.17 ± 4.68	7.406
RealNVP (w=16)	41	94.86 ± 4.02	7.476

A.3.3 Reversible backpropagation

We evaluate whether iNCA's invertibility reduces training memory while preserving task performance. Following reversible residual networks [Gomez et al., 2017], intermediate CA states are recovered through the inverse dynamics when needed for training. We save a state every 16 updates to limit numerical reconstruction errors. This reduces storage without changing the model or its training objective, although memory still increases with the number of saved states. We compare against standard backpropagation and gradient checkpointing [Chen et al., 2016], which recomputes intermediate states from saved earlier states. For the task comparison, we train four runs each for standard and reversible backpropagation on the invertible MNIST task. Evaluation at 41 CA updates shows no decrease in forward or inverse classification accuracy. This supports its use on the evaluated task, without establishing statistical equivalence.

Table 1 compares memory and runtime using the same trained weights and input batch for all three methods. Measurements use an RTX 4090, batch size 16, a 36×36 grid, and 41 CA updates in each supervised direction. We report peak allocated GPU tensor memory and median training-step time over five repetitions after two warmups. Each step includes loss computation, backpropagation, gradient clipping, and Adam. Data loading, logging, the CUDA context, and unused reserved memory are excluded. For gradient checkpointing, we test intervals of 1, 2, 4, and 8 updates and report the interval with the lowest memory use (2 updates).

Reversible backpropagation has the lowest peak memory of the tested methods, at the cost of a longer training step. Together with the paired training comparison, these results demonstrate a practical memory benefit of invertibility for iNCA training.

A.4 Additional Results

A.4.1 Latent-space scaling

For a fixed local rule, the cost of global communication grows rapidly with resolution. A 3×3 rule propagates information by at most one cell per step, so a grid of diameter D requires $\mathcal{O}(D)$ steps to connect distant cells. Updating $\mathcal{O}(D^2)$ cells at each step gives $\mathcal{O}(D^3)$ rollout work. Doubling the grid width therefore increases this cost by approximately a factor of eight. A larger fixed neighborhood improves the propagation constant but does not change this scaling.

We reduce the rollout grid by operating in a learned representation [Menta et al., 2024]. A VQ-VAE [Van den Oord et al., 2017] is first trained on Edge2Icon and then frozen. The iNCA maps between encoded edge maps and encoded icons on a 35×35 grid, after which the decoder produces a 280×280 image (fig. 10). The CA transition remains exactly invertible in latent space.

A.4.2 Per-image distributions for ambiguous MNIST

To assess the balance for individual observations beyond the aggregate split in section 5, we measure total variation (TV) distance,

$$\mathrm{TV}(y) = \frac{1}{2} \sum_{k=1}^{4} \left| \hat{p}_y(k) - p_y^*(k) \right|, \tag{42}$$

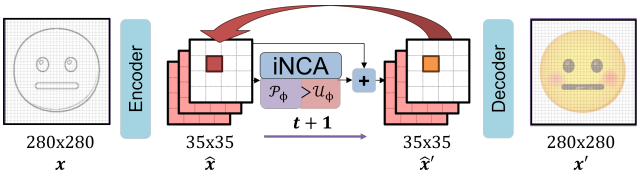

Figure 10: Latent-space iNCA. A frozen VQ-VAE maps images to a 35×35 representation, the invertible CA operates in this space, and the decoder produces the 280×280 output.

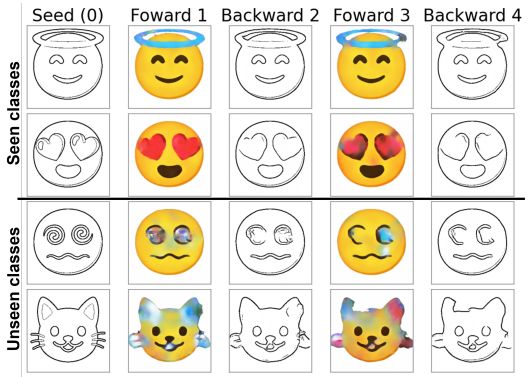

Figure 11: Forward icon generation and backward edge-map reconstruction with the latent-space iNCA. Both directions are decoded for visualization.

where $\hat{p}_y$ contains sampled label frequencies and p_y^* assigns probability $1/2$ to each valid label and zero to the others. We compute TV separately for each image before averaging. It captures both incorrect-pair predictions and imbalance between valid readings.

Table 6: Per-image distribution error on ambiguous MNIST.

Model	Parameters	Mean TV ↓
iNCA 5×5, without latent	177k	0.7400
iNCA 5×5, $d = 1$	177k	**0.1155**

Table 6 supports the benefit of latent sampling for representing the two valid readings per image. The latent model reduces distribution error, although a gap to the ideal sampling remains.

A.4.3 Continuous style variation

Beyond selecting between discrete interpretations, a latent variable can encode continuous properties that are not specified by a class seed. In the earlier direction-conditioned iNCA, holding the recovered class seed fixed while perturbing its two-dimensional style code produces digits that retain their class identity but vary in slant, stroke thickness, and curvature (fig. 13). Because this experiment predates the coupling architecture, we include it only as a qualitative illustration of the intended continuous use of z.

A.4.4 Complete Edge2Icon results

A.4.5 Bidirectional stability

All models have small noise-free round-trip errors caused by finite-precision accumulation. Their behavior differs when fresh noise is introduced at every cycle. The i-RevNets achieve the smallest round-trip errors but are less robust than the 5×5 iNCA under repeated perturbations. Accurate inversion therefore does not imply robustness to changes in the observed state.

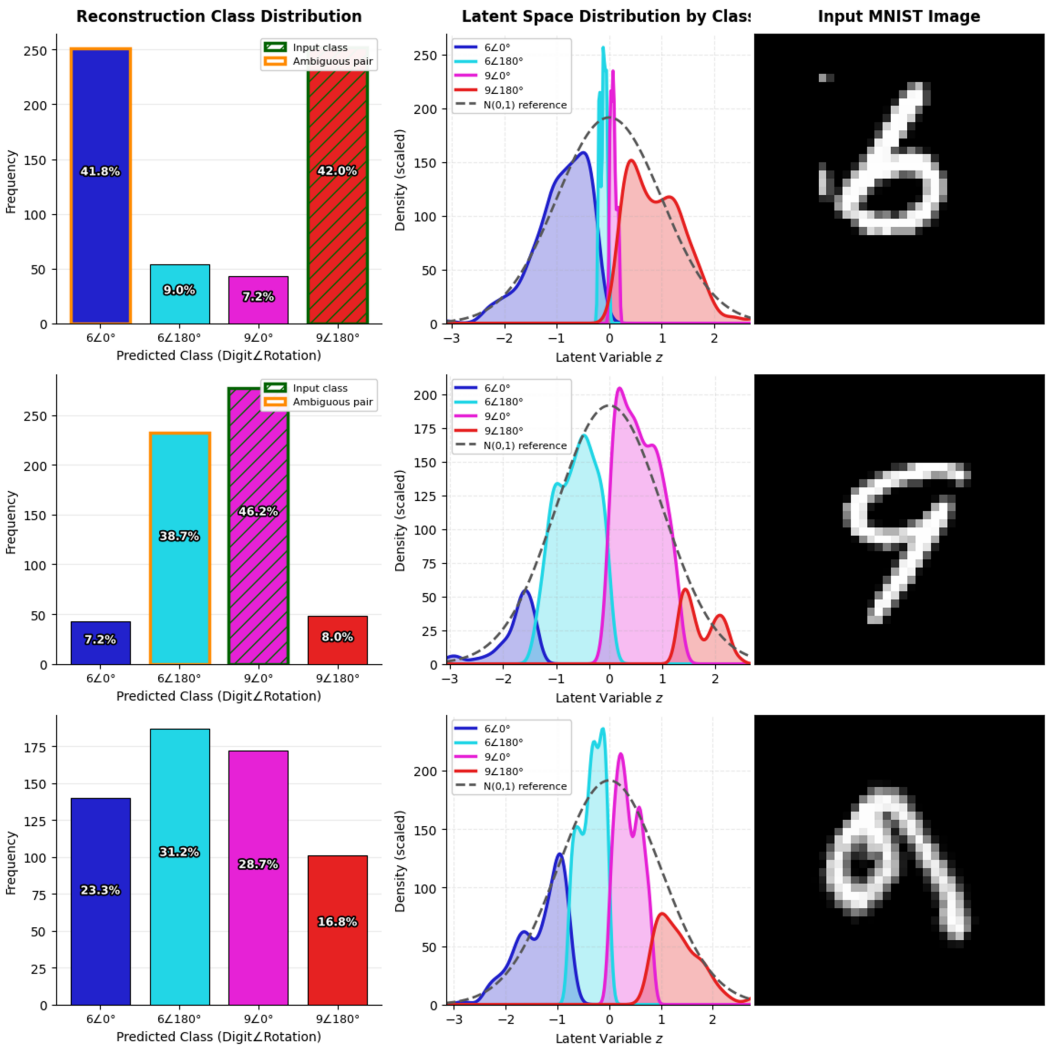

Figure 12: Enlarged version of fig. 8. Recovered classes for fixed ambiguous 6/9 observations under samples $z \sim \mathcal{N}(0, 1)$. Comparing the sampled labels across observations tests whether the model selects the corresponding admissible pair rather than an image-independent class distribution.

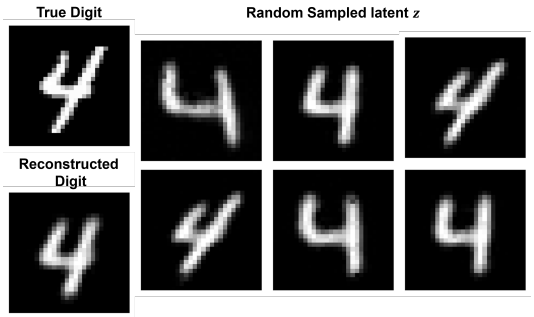

Figure 13: Qualitative MNIST style manipulation from the earlier direction-conditioned iNCA. The original digit and its reconstruction are followed by variations obtained by perturbing the two-dimensional style code while holding the class seed fixed.

Table 7: Complete Edge2Icon results on 79 training and 34 held-out emoji classes. IoU is macro-averaged over images at threshold 0.5. Values are mean ± sample standard deviation over four runs.

Model	Forward LPIPS ↓		Real-inverse IoU ↑		Projected-round-trip IoU ↑	
	Seen	Unseen	Seen	Unseen	Seen	Unseen
iNCA (Sobel)	0.0748 ± 0.0009	0.1350 ± 0.0030	0.9992 ± 0.0000	0.9716 ± 0.0023	0.9044 ± 0.0058	0.7928 ± 0.0110
iNCA (3×3)	0.0320 ± 0.0063	0.1585 ± 0.0105	0.9996 ± 0.0000	0.9727 ± 0.0023	0.9314 ± 0.0066	0.7732 ± 0.0218
iNCA (5×5)	0.0131 ± 0.0014	0.1506 ± 0.0081	0.9997 ± 0.0001	0.9654 ± 0.0014	0.9638 ± 0.0025	0.8686 ± 0.0039
iNCA ($3 \times 3 + 5 \times 5$)	0.0080 ± 0.0011	0.1485 ± 0.0118	0.9997 ± 0.0001	0.9636 ± 0.0020	0.9745 ± 0.0008	0.8749 ± 0.0097
i-RevNet (w=8)	0.0086 ± 0.0006	0.1640 ± 0.0096	0.9997 ± 0.0002	0.9630 ± 0.0050	0.9658 ± 0.0001	0.8177 ± 0.0149
i-RevNet (w=128)	0.0000 ± 0.0000	0.1489 ± 0.0051	0.9996 ± 0.0002	0.9441 ± 0.0093	0.9995 ± 0.0002	0.8399 ± 0.0108
RealNVP (w=5)	0.0057 ± 0.0030	0.1706 ± 0.0067	0.9997 ± 0.0001	0.9728 ± 0.0014	0.9742 ± 0.0062	0.7854 ± 0.0194
RealNVP (w=16)	0.0002 ± 0.0000	0.1640 ± 0.0062	0.9998 ± 0.0001	0.9698 ± 0.0045	0.9970 ± 0.0008	0.8289 ± 0.0100
Glow (w=6)	0.0015 ± 0.0003	0.1980 ± 0.0129	0.9996 ± 0.0001	0.9525 ± 0.0026	0.9870 ± 0.0006	0.7611 ± 0.0176

A.4.6 Robustness analysis

We evaluate robustness by replacing a fraction ϵ of the input pixels with black before the backward pass, sweeping ϵ from 0 to 0.9. Degradation is class-dependent: digits with distinctive geometry, such as '3' and '4', remain correct to higher values of ϵ than '8' and '9'. Seed purity decreases together with accuracy, although it is not a calibrated confidence measure.

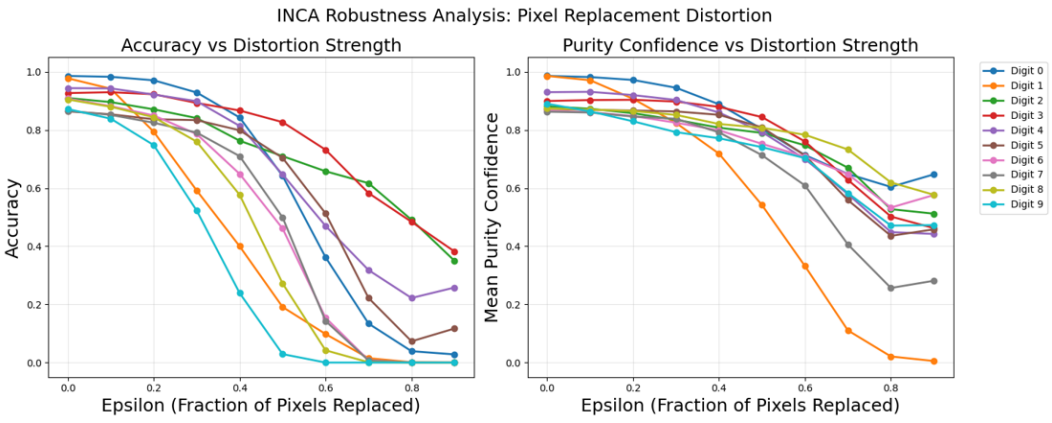

Figure 14: Backward robustness against distortion strength ϵ: per-digit accuracy for the 3×3 model.

A.5 MNIST forward classification accuracy

We evaluate generated digits with a separate supervised CNN, trained on real MNIST training images and held fixed for all evaluated models. The classifier uses three convolutional blocks with 32,

Table 8: Noise-free round-trip error and stability under repeated perturbations. Round-trip error is the mean absolute deviation between $\Phi^{-1}(\Phi(s_0))$ and s_0 after 41 steps with no injected noise. The cycle columns report how many consecutive cycles remain correct before the first failure, averaged over a maximum of 25 cycles, when fresh Gaussian noise is added after every cycle. Results are mean $\pm$ standard deviation over four runs.

Model	**Round trip** $\downarrow$	**Cycles**, σ=0.02 $\uparrow$	**Cycles**, σ=0.05 $\uparrow$
iNCA (3×3)	1.7e-5	18.2 ± 3.1	8.6 ± 2.4
iNCA (5×5)	1.6e-5	$\mathbf{22.8 \pm 1.2}$	$\mathbf{15.7 \pm 3.7}$
i-RevNet (w=8)	2.1e-6	15.9 ± 5.3	11.4 ± 6.7
i-RevNet (w=128)	**8.5e-7**	17.4 ± 2.8	12.1 ± 5.0

64, and 128 channels, followed by global average pooling and a ten-class output layer. It is trained for 40 epochs on 4,000 images per class. The classifier achieves $99.5\,\%$ accuracy on 1,000 real MNIST test images, with 100 samples per class.

A.6 The RealNVP baseline

The RealNVP-based INN [Dinh et al., 2017] is implemented with FrEIA 0.2's `RNVPCouplingBlock`. It stacks 41 independently parameterized blocks, each updating both channel groups with separate scale and translation subnetworks, with fixed random channel permutations between blocks. Each subnet uses a 3×3 convolution, ReLU, and 1×1 convolution. Hidden widths $w = 5$ give 134,644 trainable parameters. The model provides a comparison near the 5×5 iNCA's parameter budget while retaining the same depth and training settings. Log scales are tanh-clamped to magnitude 0.1 to limit multiplicative amplification. The baseline retains the 32-channel state and spatial resolution and uses the shared supervised forward/backward reconstruction objective. It is a task adaptation of RealNVP, without multiscale factorization or a likelihood objective. We report four initialization seeds per width on both MNIST and Edge2Icon.

A.7 The Glow baseline

The Glow-based INN [Kingma & Dhariwal, 2018] is implemented with FrEIA 0.2's `GLOWCouplingBlock`. It stacks 41 independently parameterized blocks, each updating both channel groups with two subnetworks that jointly predict scale and translation. Each subnet uses a 3×3 convolution, ReLU, and 1×1 convolution, with hidden width w. Coupling log scales are tanh-clamped to magnitude 0.1. On Edge2Icon, each block is preceded by a trainable channel-wise affine transformation and an LU-parameterized invertible 1×1 convolution. The affine transformation is initialized to the identity to accommodate constant seed channels. Width $w = 6$ gives 134,316 trainable parameters. Training uses 100,000 optimization steps, batch size 16, and a peak learning rate of 5×10^{-4}. As with RealNVP, this is a task adaptation without multiscale factorization or a likelihood objective.

A.8 Edge2Handbag training and evaluation

We use the paired Edge2Handbag dataset [Isola et al., 2017], resized to 64×64 without random crops or flips. RGB is premultiplied by a foreground alpha mask, obtained by marking a pixel as background when all RGB intensities exceed 220. The state has 32 channels: four edge RGBA channels, eight spatially constant latent channels, and 20 zero-initialized workspace channels. Each coupling residual uses a 5×5 perception with 128 outputs and a pointwise update network with hidden width 256. The two residuals have distinct weights, reused across 40 steps, totaling 176,928 model parameters.

Direction and objective. Let Φ denote the edge-to-handbag state rollout. Equivalently, $\Psi = \Phi^{-1}$ is the handbag-to-edge/latent direction, with generation performed by Ψ^{-1}. For a real handbag x and paired edge y, encoding starts from $[x, 0]$. Spatial averaging of the designated channels gives z_{enc}, and reconstruction starts from $[y, z_{\mathrm{enc}}, 0]$. Only the true edge and this compact code are retained; the other encoded channels are discarded. A second path generates a handbag from $[y, z, 0]$, $z \sim$

$\mathcal{N}(0, I)$, discards its hidden channels, and recovers the edge and code from visible RGBA. This combination of image and latent reconstruction follows the motivation of multimodal image-to-image methods such as BicycleGAN [Zhu et al., 2018]. Here one algebraically invertible recurrent model implements both directions.

The model objective is

$$\mathcal{L}_{\text{style}} = 10\mathcal{L}_{\text{rec}} + 5\mathcal{L}_{\text{edge}} + 0.5\mathcal{L}_{\text{zcycle}} + 0.1\,\text{MMD}^2 + \mathcal{L}_{\text{adv}}. \tag{43}$$

$\mathcal{L}_{\text{rec}}$ combines paired visible L_1 reconstruction with a range penalty of weight 0.1. $\mathcal{L}_{\text{edge}}$ averages the foreground/background-balanced edge losses for real and generated handbags. $\mathcal{L}_{\text{zcycle}}$ is the L_1 error between every recovered latent plane and its sampled constant value; it is evaluated after hidden-channel removal. MMD matches the pooled real-image codes to Gaussian samples using the kernel convention in section A.3.1. An edge-conditioned Wasserstein critic supervises the random-code images, with gradient-penalty weight 10 and adversarial weight 1. No separate colour-diversity loss is enabled. Training uses 250k steps, batch size 32, Adam with learning rate 10^{-4} and betas $(0.5, 0.95)$, 2k warmup steps, and learning-rate decay over the final tenth of training. The critic starts at step 5k; the learning rate is warmed up over the following 1k steps.

Table 9: Additional Edge2Handbag diagnostics on all 200 validation pairs, with eight Gaussian samples per edge. Mean $\pm$ sample standard deviation across the four saved runs. These measurements characterize the evaluated configuration; no comparison with other generators is made.

Model	Reconstruction LPIPS $\downarrow$	Diversity LPIPS $\uparrow$	Real-inverse IoU $\uparrow$	Projected round trip IoU $\uparrow$
iNCA 5×5, $d = 8$	0.1813 ± 0.0044	0.2143 ± 0.0026	0.5443 ± 0.0015	0.9986 ± 0.0002

Table 10: Appearance-code controls on the same validation pairs. Reconstruction LPIPS uses the corresponding encoded code, a code from a different validation handbag, or zero. Pooled code MAE compares Gaussian samples with codes recovered from generated visible RGBA.

Run	Encoded LPIPS $\downarrow$	Shuffled LPIPS $\downarrow$	Zero LPIPS $\downarrow$	Pooled z MAE $\downarrow$
A	0.1770	0.2764	0.2493	0.2828
B	0.1795	0.2789	0.2436	0.2602
C	0.1874	0.2836	0.2499	0.2529
D	0.1813	0.2753	0.2344	0.3439