E(n)-Equivariant Graph Neural Cellular Automata

Gennaro Gala [1] **Daniele Grattarola** [2] **Erik Quaeghebeur** [1]

Abstract

Cellular automata (CAs) are computational models exhibiting rich dynamics emerging from the local interaction of cells arranged in a regular lattice. Graph CAs (GCAs) generalise standard CAs by allowing for arbitrary graphs rather than regular lattices, similar to how Graph Neural Networks (GNNs) generalise Convolutional NNs. Recently, Graph Neural CAs (GNCAs) have been proposed as models built on top of standard GNNs that can be trained to approximate the transition rule of any arbitrary GCA. Existing GNCAs are anisotropic in the sense that their transition rules are not equivariant to translation, rotation, and reflection of the nodes' spatial locations. However, it is desirable for instances related by such transformations to be treated identically by the model. By replacing standard graph convolutions with E(n)-equivariant ones, we avoid anisotropy by design and propose a class of isotropic automata that we call E(n)-GNCAs. These models are lightweight, but can nevertheless handle large graphs, capture complex dynamics and exhibit emergent self-organising behaviours. We showcase the broad and successful applicability of E(n)-GNCAs on three different tasks: (i) pattern formation, (ii) graph autoencoding, and (iii) simulation of E(n)-equivariant dynamical systems.

1. Introduction

The design of collective intelligence, i.e. the ability of a group of simple agents to collectively cooperate towards a unifying goal, is a growing area of machine learning research aimed at solving complex tasks through *emergent computation* (Ha & Tang, 2022). The interest in these techniques stems from their striking similarity to real biological systems—such as insect swarms and bacteria colonies—and from their natural scalability as distributed systems (Mitchell, 2009).

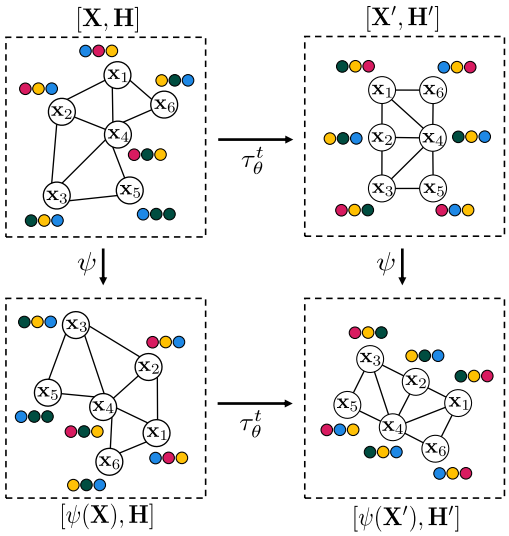

Figure 1. E(n)-GNCA commutative diagram: For any number of time steps t transition rule τ_θ is run, output coordinates $\mathbf{X}'$ and output node features $\mathbf{H}'$ are respectively E(n)-equivariant and E(n)-invariant to rigid transformations of input coordinates $\mathbf{X}$. Node features are represented with 3 colored dots attached to each node.

Cellular automata (CAs) (von Neumann, 1963) represent a natural playground for studying collective intelligence and morphogenesis (shape-forming processes), because of their discrete-time and Markovian dynamics (Turing, 1990). CAs are computational models inspired by the biological behaviors of cellular growth. As such, they are capable of producing complex emergent *global* dynamics from the iterative, possibly asynchronous application of *localized* transition rules (aka update rules), that can but do not need to have an analytical formulation (Adamatzky, 2010).

Research on applying neural nets for learning and designing CA rules can be traced back to Wulff & Hertz (1992), with subsequent notable contributions by Elmenreich & Fehérvári (2011), Nichele et al. (2017), and Gilpin (2019). Recently, Neural Cellular Automata (NCAs) have been proposed as CAs with transition rules encoded as—typically light-weight—neural networks. They have been successfully applied for designing self-organizing systems for morphogenesis in 2D and 3D (Mordvintsev et al., 2020; Sudhakaran et al., 2021), image generation and classification (Palm et al., 2022; Randazzo et al., 2020), and reinforcement learning

[1]Eindhoven University of Technology, The Netherlands [2]Independent researcher. Correspondence to: Gennaro Gala <g.gala@tue.nl>.

Preprint.

(Huang et al., 2020). This line of work has a common theme: It assumes a fixed discrete geometry for the CA cells, which are typically arranged in n-dimensional, equispaced, and oriented lattices.

Subsequently, Grattarola et al. (2021) introduced GNCAs (Graph NCAs) by extending NCAs to the general setting of graphs, and showed that Graph Neural Networks are natural and universal engines for learning any desired transition rule. However, their formulation does not consider the possible symmetries in the state space, instead relying on a fixed frame of reference even for states representing spatial information like position and velocity. Further, their architecture does not allow nodes to have hidden states, which have been proven to be useful for perception and for keeping track of evolution history (Mordvintsev et al., 2020).

By building on Satorras et al.'s (2021b) work on E(n)-equivariant Graph Neural Networks (EGNNs), we overcome these limitations and present GNCAs that respect isometries in the state space *by design*. We name the resulting class of models E(n)-GNCAs. In section 4, we showcase the broad, successful applicability of E(n)-GNCAs on three different tasks: (i) pattern formation, (ii) graph auto-encoding, and (iii) simulation of E(n)-equivariant dynamical systems.

2. Preliminaries and Related Work

In this section, we introduce necessary concepts of and relevant prior work on graphs, cellular automata, and (equivariant) graph neural networks. These support and contextualize the definition of the class of models we propose.

Graphs A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of an unordered set of nodes $\mathcal{V} = \{1, \dots, |\mathcal{V}|\}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Its neighbourhood function $\mathcal{N}$ is defined for every node $i \in \mathcal{V}$ by $\mathcal{N}(i) = \{j \in \mathcal{V} : (i,j) \in \mathcal{E}\}$. A graph can be equivalently defined with an adjacency matrix $A \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where A_{ij} is 1 if and only if $(i,j) \in \mathcal{E}$.

We can attach a state $\mathbf{s}_i \in \mathcal{S}$ to each node i and an attribute $\mathbf{e}_{ij} \in \mathcal{A}$ to each edge (i,j), where for now we leave the state space $\mathcal{S}$ and attribute set $\mathcal{A}$ unspecified. A node state $\mathbf{s}_i$ typically consists of components such as location $\mathbf{x}_i$, velocity $\mathbf{v}_i$, and node features $\mathbf{h}_i$. Jointly for all nodes or edges, we write $\mathbf{S}$—with components $\mathbf{X}$, $\mathbf{V}$, and $\mathbf{H}$—and $\mathbf{E}$, which implicitly carry with them the underlying graph.

2.1. Graph (Neural) Cellular Automata

Graph Cellular Automata A *Graph Cellular Automaton* (GCA) is a triple $(\mathcal{G}, \mathcal{S}, \tau)$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a graph and $\mathcal{S}$ is a discrete or continuous state space. The map $\tau : \mathcal{S} \times 2^{\mathcal{S}} \to \mathcal{S}$ is used as a *local* transition rule to update the state $\mathbf{s}_i \in \mathcal{S}$ of each of the graph's nodes $i \in \mathcal{V}$ as a function of its current state and its neighbour's states:

$$\mathbf{s}_i' = \tau\big(\mathbf{s}_i, \{\mathbf{s}_j : j \in \mathcal{N}(i)\}\big). \tag{1}$$

We compactly write $\mathbf{S}' = \tau(\mathbf{S})$ to indicate the synchronous application of τ to all nodes. Standard CAs—like elementary CAs (Wolfram, 2018) and Conway's Game of Life (Adamatzky, 2010)—use a grid for the graph, have integer-valued locations $\mathbf{x}_i \in \mathbb{Z}^n$ and use a single binary value for their features $\mathbf{h}_i$.

Anisotropy & Isotropy Of great importance for CAs are the properties *anisotropy* and *isotropy*: The former implies being directionally dependent, as opposed to the latter, which indicates homogeneity in all directions. Specifically, anisotropic transition rules are *not* invariant to rotations, translations and reflections of the states thus resulting in nodes being oriented in a specific direction and prohibiting the existence of differently oriented states of interest (Mordvintsev et al., 2022; Grattarola et al., 2021). In contrast, isotropy allows transition rules to act similarly regardless of how the nodes are oriented, thus allowing proper design of self-organising (and living) systems.

Neural Cellular Automata A neural cellular automaton (NCA) is a light-weight neural net with parameters θ representing a parameterised transition rule τ_θ (Mordvintsev et al., 2020). In this setting, states are represented as typically low-dimensional vectors and the differentiability of the transition rule allows to optimise its parameters θ via backpropagation through time (Lillicrap & Santoro, 2019). Recent work has shown the successful application of deep learning techniques for NCAs, showing that neural transition rules can be efficiently learned to exhibit complex desired behaviors (Mordvintsev et al., 2020; 2022; Tesfaldet et al., 2022; Grattarola et al., 2021; Palm et al., 2022).

As already pointed out by Tesfaldet et al. (2022), NCAs are not structurally equivalent to (deep) feed-forward neural nets, where a *acyclic* directed computation graph induces a *finite* impulse response. Instead, NCAs can be viewed as Recurrent Neural Networks (RNNs) (Rumelhart et al., 1985), where a *cyclic* directed computation graph induces an *infinite* impulse response, enabling feedback and time-delayed interactions. Notably, RNNs and CAs—and therefore NCAs—are known to be Turing complete (Pérez et al., 2019; Rendell, 2002).

2.2. Graph Neural Networks

Graph Neural Networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008) have become the go-to method for representation learning on graphs. The core functionality of GNNs is the message-passing scheme. Let $\mathbf{s}_i \in \mathbb{R}^s$ represent the feature vector of node i and $\mathbf{e}_{ij} \in \mathbb{R}^e$ the (possibly available) feature vector of edge (i,j). A message-passing layer

updates the features of node i as follows:

$$\mathbf{s}_i' = \gamma(\mathbf{s}_i, \bigoplus_{j \in \mathcal{N}(i)} \phi(\mathbf{s}_i, \mathbf{s}_j, \mathbf{e}_{ji})), \tag{2}$$

where ϕ is the message function, $\bigoplus$ is a permutation-invariant operation to aggregate the set of incoming messages (usually a sum or an average), and γ is the node update function. The operators ϕ, $\bigoplus$, and γ must all be differentiable, allowing message-passing layers to be stacked sequentially and then optimised end-to-end with stochastic gradient descent.

2.3. E(*n*)-Equivariant Graph Neural Networks

Our work builds on E(n)-Equivariant GNNs (EGNNs) (Satorras et al., 2021b). In this setting, every graph node i has coordinates $\mathbf{x}_i \in \mathbb{R}^n$ and node features $\mathbf{h}_i \in \mathbb{R}^h$, and an edge $(i, j) \in \mathcal{E}$ can possibly have attributes $\mathbf{e}_{ij} \in \mathbb{R}^e$. EGNNs represent a class of GNNs explicitly designed to be permutation equivariant with respect to the nodes (like any GNN), and translation, rotation and reflection equivariant with respect to nodes' coordinates. The isometry group corresponding to these symmetries acting in an n-dimensional Euclidean space is called the Euclidean group E(n). We will formally discuss the *key features* of EGNNs (cf. Equation 12) while presenting our method in section 3.

E(*n*)-Equivariant Graph Convolutions Given a graph $\mathcal{G}$, node coordinates $\{\mathbf{x}_i\}$, node features $\{\mathbf{h}_i\}$ and *optional* edge attributes $\{\mathbf{e}_{ij}\}$ an E(n)-Equivariant Graph Convolution (EGC) sequentially performs:

$$\mathbf{m}_{ij} = \phi_m(\|\mathbf{x}_i - \mathbf{x}_j\|^2, \mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij}) \tag{3}$$

$$\mathbf{x}_i' = \mathbf{x}_i + \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} (\mathbf{x}_i - \mathbf{x}_j)\phi_x(\mathbf{m}_{ij}) \tag{4}$$

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \tag{5}$$

$$\mathbf{h}_i' = \phi_h(\mathbf{h}_i, \mathbf{m}_i) \tag{6}$$

where $\phi_m : \mathbb{R}^{2h+e+1} \to \mathbb{R}^m$, $\phi_x : \mathbb{R}^m \to \mathbb{R}^1$ and $\phi_h : \mathbb{R}^{h+m} \to \mathbb{R}^{h'}$ are dense neural networks. Concisely, we write $\mathbf{X}', \mathbf{H}' = \text{EGC}(\mathbf{X}, \mathbf{H}, \mathbf{E})$. Note that when $h' = h$ we can use a skip connection in Equation 6 as follows:

$$\mathbf{h}_i' = \phi_h^+(\mathbf{h}_i, \mathbf{m}_i) = \phi_h(\mathbf{h}_i, \mathbf{m}_i) + \mathbf{h}_i. \tag{7}$$

Skip connections have been proven to improve model resilience to vanishing gradients and over-smoothing (Zhao & Akoglu, 2019).

E(*n*)-Equivariant Graph Convolutions with Attention In order to give the model the freedom to assign different weights when aggregating messages, we can use attention weights and replace equation 5 with:

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \phi_a(\mathbf{m}_{ij})\mathbf{m}_{ij} \tag{8}$$

where $\phi_a : \mathbb{R}^m \to [0, 1]$[1] is a dense neural network that takes a message $\mathbf{m}_{ij}$ as input and outputs its attention weight $\phi_a(\mathbf{m}_{ij})$. Attention weights are particularly advantageous when a fully connected graph is used (Satorras et al., 2021b; Vaswani et al., 2017).

E(*n*)-Equivariant Graph Convolutions with Velocity When nodes represent bodies with velocities, we can extend the previous formulation to explicitly take velocities into account. Specifically, given node velocities $\{\mathbf{v}_i\}$ we can replace the coordinate update in Equation 4 with the following two steps:

$$\mathbf{v}_i' = \phi_v(\mathbf{h}_i, \|\mathbf{v}_i\|)\mathbf{v}_i + \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} (\mathbf{x}_i - \mathbf{x}_j)\phi_x(\mathbf{m}_{ij}) \tag{9}$$

$$\mathbf{x}_i' = \mathbf{x}_i + \mathbf{v}_i' \tag{10}$$

where $\phi_v : \mathbb{R}^{h+1} \to \mathbb{R}^1$ is a dense neural network. Without affecting equivariance, and different from Satorras et al. (2021b), we input $\|\mathbf{v}_i\|$ (and not only $\mathbf{h}_i$) to ϕ_v since we found it to be very beneficial in practice (cf. subsection 4.3). Concisely, we write $\mathbf{X}', \mathbf{V}', \mathbf{H}' = \text{EGC}(\mathbf{X}, \mathbf{V}, \mathbf{H}, \mathbf{E})$.

E(*n*)-Equivariant Graph Neural Networks An E(n)-Equivariant GNN is a stack of $\ell \geq 1$ EGCs applied sequentially. Concisely, we write $\mathbf{X}', \mathbf{H}' = \text{EGNN}_\ell(\mathbf{X}, \mathbf{H}, \mathbf{E})$ to denote the application of an EGNN with ℓ layers.

3. E(*n*)-equivariant Graph Neural CAs

Our work builds on the connection between isotropic GCAs and EGNNs. This can be seen by comparing Equation 1 with Equations 4 and 6. Specifically, we consider a setting in which a *parametrised* transition rule is implemented with a single E(n)-equivariant Graph Convolution (EGC) acting on a continuous state space $\mathcal{S} \equiv \mathbb{R}^{n+h}$, or $\mathcal{S} \equiv \mathbb{R}^{2n+h}$ when velocity is included. A layered EGNN is also a possible and viable approach to modeling transition rules, especially if one wants to account for higher-order neighbours when performing a single state update (Chan, 2019).

We introduce E(n)-Equivariant Graph Neural Cellular Automata, E(n)-GNCAs for short. They use a single EGC for the parameterised transition rule τ_θ. Similarly to plain cellular automata, τ_θ is repeatedly applied over time:

$$\mathbf{X}', \mathbf{H}' = \tau_\theta^t([\mathbf{X}, \mathbf{H}]) = \underbrace{\tau_\theta \circ \cdots \circ \tau_\theta}_{t \text{ times}}([\mathbf{X}, \mathbf{H}]), \tag{11}$$

where $\mathbf{X}$ represents input node coordinates and $\mathbf{H}$ represents input node features. Note that (i) to avoid clutter in Equation 11 we did not consider possibly available edge attributes $\mathbf{E}$, (ii) a similar formulation is possible when velocities $\mathbf{V}$ are available (cf. Equation 9), and (iii) the dependency of τ_θ on a static graph $\mathcal{G}$ is left implicit in order to keep notation uncluttered. The overall state configuration $\mathbf{S}$ of an E(n)-GNCA is defined as $\mathbf{S} = [\mathbf{X}, \mathbf{H}]$, or $\mathbf{S} = [\mathbf{X}, \mathbf{H}, \mathbf{V}]$ when velocity is available, and consequently we denote the t-times application of the model transition rule as $\mathbf{S}' = \tau_\theta^t(\mathbf{S})$. Importantly, the transition rules we consider are 1-step Markovian, meaning that automaton state at step $t+1$ is fully determined by the state at step t.

E(n)-equivariance, E(n)-invariance and Isotropy Analogously to plain EGNNs (Satorras et al., 2021b), for any positive integer $t \in \mathbb{N}^+$, orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ and translation vector $b \in \mathbb{R}^n$, our neural transition rule τ_θ satisfies the following:

$$\psi(\mathbf{X}'), \mathbf{H}' = \tau_\theta^t([\psi(\mathbf{X}), \mathbf{H}]), \tag{12}$$

where $\mathbf{X}', \mathbf{H}' = \tau_\theta^t([\mathbf{X}, \mathbf{H}])$ and $\psi(\mathbf{X}) = Q\mathbf{X} + b$ is shorthand for $(Q\mathbf{x}_1 + b, \dots, Q\mathbf{x}_{|\mathcal{V}|} + b)$. The map ψ is a *rigid transformation* (aka *isometry*), and represents a rotation-reflection-translation of the coordinates. As such, ψ preserves the Euclidean distance between every pair of nodes. As illustrated with the commutative diagram in Figure 1, applying ψ to input coordinates $\mathbf{X}$ and then running transition rule τ_θ^t will give the same results as first running τ_θ^t and then applying ψ to $\mathbf{X}'$. Thus, output coordinates $\mathbf{X}'$ and output node features $\mathbf{H}'$ are respectively E(n)-equivariant and E(n)-invariant to rigid transformations of input coordinates $\mathbf{X}$. Intuitively, these properties are a consequence of the model only processing relative distances and never being aware of absolute node locations (cf. Equation 3 and Equation 4).[1] Importantly, the E(n)-invariance of the node features and the E(n)-equivariance of the node coordinates make E(n)-GNCAs isotropic *by design*.

Hidden States & Perception Similarly to Mordvintsev et al. (2020; 2022), Palm et al. (2022), and Chan (2019), but different from Grattarola et al. (2021), our model has the necessary inductive bias for modelling hidden states, as it offers location-independent node features $\mathbf{H}$. As Mordvintsev et al. (2020), we interpret hidden states as a signal mechanism for orchestrating morphogenesis: All nodes share the same genome, i.e. the transition rule, and only differ from the information encoded by the signaling they receive, emit, and store internally, i.e. their node features. In case node features $\mathbf{H}$ are *not* available in advance, we can either set them to $\mathbf{1}$ or randomly initialize them, and give the model the freedom to learn and use them while evolving. Further, messages $\{\mathbf{m}_{ij}\}$ (cf. Equation 3) are similar in spirit to the perception vectors of Mordvintsev et al. (2020; 2022), as they encode what nodes perceive of the environment from communicating with their neighbors.

Given (i) the interest in what would happen as $t \to \infty$ and (ii) the recurrent architecture of our model, we normalise node feature $\mathbf{H}$ after each transition rule application so as to mitigate problems like over-smoothing, exploding/vanishing gradients, and training instabilities. Specifically, after every transition rule application, we normalise node features $\mathbf{H}$ with either PairNorm (Zhao & Akoglu, 2019) or NodeNorm (Zhou et al., 2021), helpful *parameter-free* normalisation techniques for deep GNNs. Further, we use the hyperbolic tangent $\mathsf{TanH}()$ as non-linear activation function—a common design choice in RNNs (Lipton et al., 2015)—and the skip connection defined in Equation 7, which has proven to be very beneficial for deep GNNs (Zhao & Akoglu, 2019).

Global propagation from local interactions Message-passing GNNs require ℓ layers to allow communication between nodes that are ℓ-hops away. Several tasks in graph ML tend to be very challenging when the diameter of the underlying graph $\mathcal{G}$ is larger than the number of layers used, and that is because the receptive field of the network may not comprise the whole graph (Zhao & Akoglu, 2019; Alon & Yahav, 2021). Further, to avoid severe over-smoothing (Li et al., 2018), most popular GCN-style networks (Kipf & Welling, 2017) tend to be shallow, with narrow receptive fields, leading to *under-reaching* (Wenkel et al., 2022). To avoid this limitation, it is common to exchange messages among all nodes and providing the edge information $(i, j) \in \mathcal{E}$ as a boolean flag within the edge attributes (Liu et al., 2019; Satorras et al., 2021b;a). This is computationally quadratic in the number of nodes, and therefore very expensive and challenging when processing large graphs.

In our setting, due to the 1-step Markovian property of τ_θ, the effective receptive field of E(n)-GNCA localized message-passing grows larger with each state update until eventually encompassing the whole graph. In this way global propagation of information arises from spatially localized interactions of nodes. In other words, iterative local message-passing circumvents the quadratic complexity and related challenges of exchanging messages among all nodes at each step. This self-organizing process does not require any external control or centralized leader: nodes communicate with their neighbors to make collective decisions about the final configuration of the nodes. This globally consistent and complex behaviour, which arises from strictly local interactions, is a particular feature of (N)CAs as we show in our experiments. Finally, we emphasise that—despite the localized computation—we are allowed to express global information within the loss function employed.

[1] We refer the reader to the appendix of Satorras et al. (2021b) for a formal proof of the equivariance/invariance of EGNNs.

4. Experiments

We showcase the broad and successful applicability of E(n)-GNCAs in three different tasks: (i) pattern formation, (ii) graph autoencoding and (iii) simulation of E(n)-Equivariant dynamical systems. We set $h = 16$ (hidden state dimension) and $m = 32$ (message dimension) throughout *all* experiments which lead to an overall automaton size of only 5K parameters *irrespective* of the coordinate dimension n being used. Our code is available at github.com/gengala/egnca. We are grateful to the developers of the main software packages used for this work: Pytorch (Paszke et al., 2019), PyTorch Geometric (Fey & Lenssen, 2019) and PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019).

4.1. Pattern Formation

Inspired by prior work on CA morphogenesis (Mordvintsev et al., 2020; 2022; Grattarola et al., 2021), we show how E(n)-GNCAs can be trained to converge to a given fixed target state. In our case, the target is a sparse geometric graph $\mathcal{G}$ that visually defines a recognisable 2D or 3D shape. Specifically, the goal is to learn a transition rule τ_θ that *morphs* randomly initialised coordinates $\overline{\mathbf{X}}$ to a given target point cloud $\hat{\mathbf{X}}$ by convolving over $\mathcal{G}$ and assuming a prior 1-to-1 correspondence between nodes in $\overline{\mathbf{X}}$ and $\hat{\mathbf{X}}$.

E(n)-invariant objective Contrary to Grattarola et al. (2021), we are *not* interested in a specific orientation of $\hat{\mathbf{X}}$ and therefore we do *not* optimise the model by minimising the MSE between coordinates reached by the model and target coordinates, i.e. $\|\mathbf{X}' - \hat{\mathbf{X}}\|^2$ where $[\mathbf{X}', \mathbf{H}'] = \tau_\theta^t([\overline{\mathbf{X}}, \mathbf{1}])$. The former, moreover, would *not* be a suitable objective for our automata since it accounts for specific locations whereas our model only uses relative distances during its computation (cf. Equations 3 and 4). Therefore, for every pair of nodes $(i, j) \in \mathcal{V} \times \mathcal{V}$, we minimise the MSE between their distance in the model's final configuration and the target one. Formally, we define an E(n)-invariant objective defined as follows:

$$\mathcal{L}_{\mathsf{INV}} = \frac{1}{|\mathcal{V}|^2} \sum_{(i,j)\in\mathcal{V}\times\mathcal{V}} (\|\mathbf{x}'_i - \mathbf{x}'_j\| - \|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|)^2. \quad (13)$$

This objective (cf. Equation 13) provides a weaker supervision signal than the one by Grattarola et al. (2021), therefore leading to a much more challenging task. That is because for Grattarola et al. (2021) every node has only a single constraint to satisfy, i.e. being close to a specific global location, whereas in our case every node has $|\mathcal{V}| - 1$ constraints to satisfy, i.e. its distances w.r.t. all the other nodes in $\mathcal{G}$, *not* only its neighbors. Interestingly, optimizing for Equation 13 results in learning a transition rule τ_θ such that $[\mathbf{X}', \mathbf{H}'] = \tau_\theta^t([\overline{\mathbf{X}}, \mathbf{1}])$ and $\mathbf{X}' = \psi(\hat{\mathbf{X}})$ for any arbitrary rigid transformation $\psi(\cdot)$. In other words, our objective gives the model the freedom to converge in any possible orientation of the target. In practice, one could avoid evaluating $\mathcal{O}(|\mathcal{V}|^2)$ distances by only considering a randomly sampled subset of edges when computing Equation 13. Our objective is similar in spirit to the one by Mordvintsev et al. (2022), where a rotation-reflection invariant objective is used in the image domain. Similarly, the loss function we employ is an E(n)-invariant point cloud description and fits well with the isotropy of our model.

Training We mostly follow the experimental setup in (Mordvintsev et al., 2020; Grattarola et al., 2021). First, we create a large pool (aka *cache*) of K states $\{\mathbf{S}^{(k)}\}_{k=1}^K = \{[\mathbf{X}^{(k)}, \mathbf{H}^{(k)}]\}_{k=1}^K$, each initialised as $[\overline{\mathbf{X}}, \mathbf{1}]$, where $\overline{\mathbf{X}} \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{1})$. Then, we randomly sample a mini-batch from the pool and use it as input to transition rule τ_θ, which runs for a number of time steps t sampled uniformly from the interval $[15, 25]$[2]. Once a mini-batch is processed, we apply backpropagation through time (BPTT) (Lillicrap & Santoro, 2019) to update parameters θ according to Equation 13. To promote persistency, we use the pool as a *replay memory*, so that, once an optimisation step is performed, we replace the pool state $\mathbf{S}^{(k)}$ with $\tau_\theta^t(\mathbf{S}^{(k)})$ for every $\mathbf{S}^{(k)}$ in the current mini-batch. This allows next training iterations to account for states that already result from a repeated application of the transition rule, thus encouraging the model to persist in the target state after reaching it. Further, before processing a mini-batch, the state with the highest loss value is replaced with the initial state $[\overline{\mathbf{X}}, \mathbf{1}]$ so as to both stabilise training and, more importantly, avoid catastrophic forgetting. Finally, to also promote regeneration, we perturb half of the point clouds in the batch by adding Gaussian noise. Specifically, one quarter is perturbed globally and another only locally.

Results We consider the following geometric graphs available in PyGSP (Defferrard et al., 2017): a regular 2D grid (256 nodes), a 3D torus (256 nodes) and the Stanford bunny (2503 nodes). Figure 2 shows (part of) E(n)-GNCA trajectories as well as the loss value (cf. Equation 13) w.r.t. the coordinates at each time step shown. Remarkably, our model learns to converge to a stable attractor of the given geometric graph after any number of time steps $t > 15$. Furthermore, the model exhibits regeneration abilities by being robust against perturbations of the coordinates. More experimental details and results can be found in Appendix A.

4.2. Graph autoencoding with Cellular Automata

In this section, we show how E(n)-GNCAs can be deployed as performant Graph AutoEncoders (GAEs) (Kipf

[2]The interval considered represents a trade-off between computational complexity, stability during training, and a sufficient number of time steps to allow the model to learn dynamical patterns for the desired behavior.

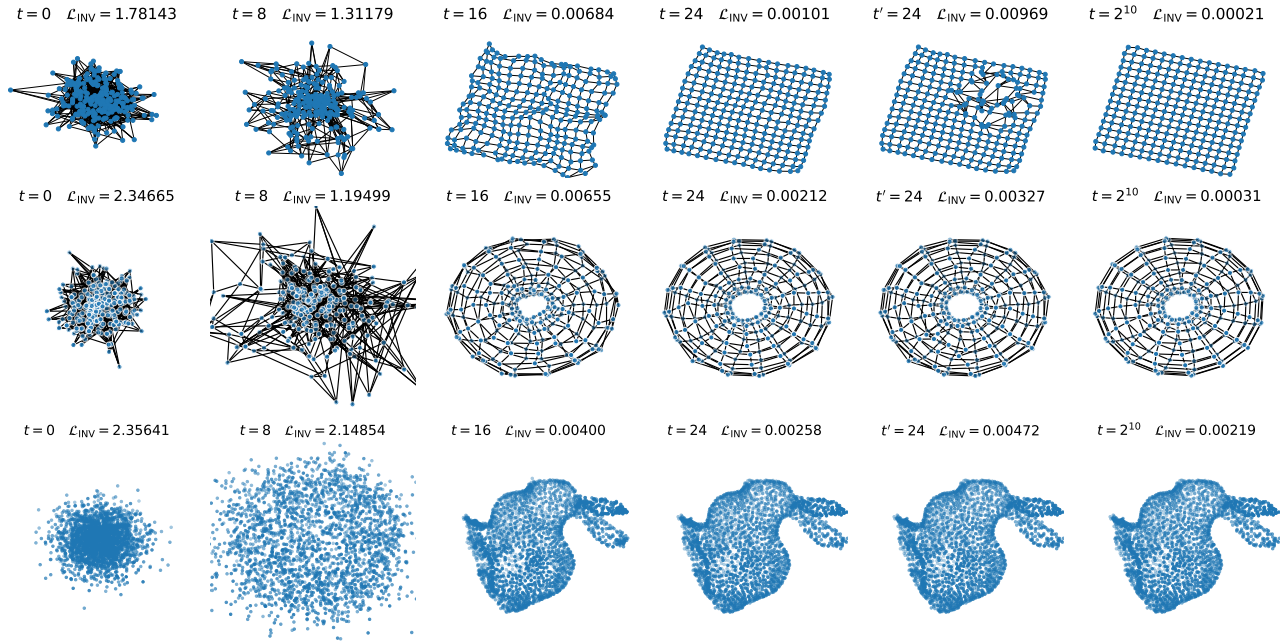

Figure 2. E(n)-GNCA convergence to a 2D grid (top), a 3D torus (middle) and the Stanford geometric bunny (bottom). The first 4 columns show E(n)-GNCA states at different time steps. The second to last column show either a local or global damage of coordinates at $t = 24$. Finally, the last column aims to both show regeneration and persistency abilities by running the transition rule for 1000 extra time steps after perturbation has occurred. We report the loss value of Equation 13 for the state in each figure. The nearest-neighbor edges of the Stanford bunny are not shown so as to avoid clutter. We report complete trajectories in Appendix A. Best viewed digitally and zoomed in.

& Welling, 2016), despite their single-layered architecture and recurrent computation. In graph autoencoding one has available a set of (possibly featureless) graphs $\{\mathcal{G}_n\}$ and one wants to learn node representations that can be used to reconstruct the underlying ground-truth adjacency matrices (Satorras et al., 2021b; Liu et al., 2019). For this task, we report more details and additional results in Appendix B.

Datasets We consider five datasets of featureless graphs of varying size, connectivity and properties: COMM-S (100 graphs, 2 communities, 12–20 nodes) (Liu et al., 2019), PLANAR-S (200 planar graphs, 12–20 nodes), PLANAR-L (200 planar graphs, 32–64 nodes), SBM (200 stochastic block model graphs, 2–5 communities, 44–187 nodes) (Martinkus et al., 2022) and PROTEINS (918 graphs, 100–500 nodes) (Dobson & Doig, 2003). Figure B.5 shows some examples of such graphs. We split all datasets into training (80%), validation (10%) and test (10%).

Training For each training graph $\mathcal{G}_n$ we create a small pool of K states $\{[\mathbf{X}^{(n,k)}, \mathbf{H}^{(n,k)}]\}$. Every $\mathbf{H}^{(n,k)}$ is again initialised as $\mathbf{1}$ whereas input node coordinates $\mathbf{X}^{(n,k)}$ now follow an isotropic Gaussian $\mathcal{N}(\mathbf{0}, \sigma\mathbf{1})$.[3] As such, the model can be viewed a generative model *conditioned* on $\mathcal{G}$. A mini-batch is now created by first considering a random subset of training graphs and then sampling a random pool state each. Every mini-batch state $\mathbf{S}^{(n,k)}$ is then run by τ_θ for $t \in [t_1, t_2]$ random time steps eventually reaching state $[\mathbf{X}', \mathbf{H}'] = \tau_\theta^t(\mathbf{S}^{(n,k)})$. Finally, we apply an E(n)-invariant decoding scheme based on distances between nodes $\mathbf{X}'$ so that the reconstructed soft adjacency matrix $\hat{A} \in [0,1]^{|\mathcal{V}|\times|\mathcal{V}|}$ is defined as:

$$\hat{A}_{ij} = \frac{1}{1 + \exp(\delta_2(\|\mathbf{x}'_i - \mathbf{x}'_j\|_2^2 - \delta_1))} \in [0,1], \qquad (14)$$

where δ_1 and δ_2 are learnable positive scalar parameters. The model is trained by minimising the binary cross-entropy (BCE) between the ground-truth adjacency A and the predicted soft one $\hat{A}$, namely:

$$\mathcal{L}_{\text{BCE}} = -\sum_{ij} A_{ij} \ln(\hat{A}_{ij}) + (1 - A_{ij}) \ln(1 - \hat{A}_{ij}). \qquad (15)$$

Furthermore, we require our autoencoders to be persistent, which means autoencoding has to be possible from $\mathbf{X}'$ for any $t > t_1$. To promote persistency, we use a *multi-target* replay strategy. Specifically, after every optimisation step, we replace the reached state $[\mathbf{X}', \mathbf{H}']$ with the pool state that originated it, and randomly re-initialise pool states after a given number of maximum replacements so as to avoid catastrophic forgetting.

[3] Injecting Gaussian noise as initial node features has originally been proposed by Liu et al. (2019), and then also used as a way of overcoming the symmetry problem and over-smoothing (Satorras et al., 2021b; Sato et al., 2021; Godwin et al., 2022).

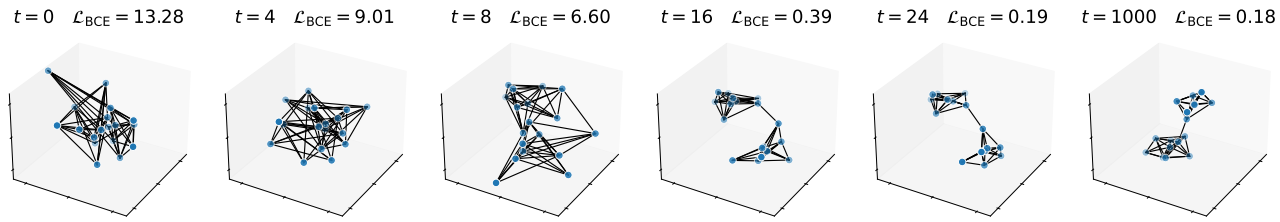

Figure 3. E(n)-GNCA coordinates at different time steps for a test-set graph in COMM-S. In each figure, we plot the ground-truth edges and report the binary cross-entropy (cf. Equation 15). Best viewed digitally and zoomed in.

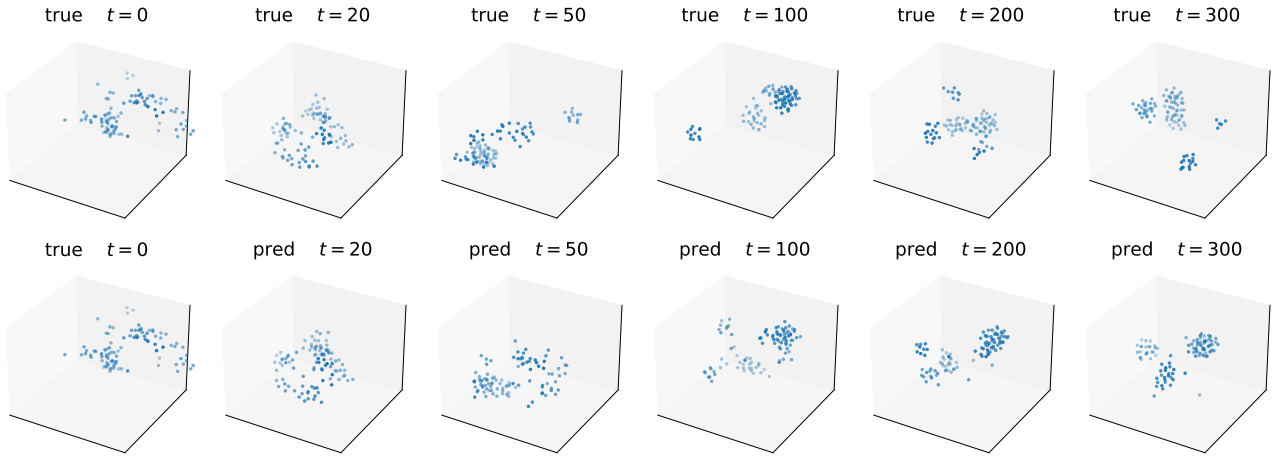

Figure 4. Boids simulation. First (Second) row shows a ground-truth (predicted) trajectory at different time steps. E(n)-GNCA learns a flocking behaviour similar to the target system, although with smoother and less precise trajectories.

A 3D demo In a first demo experiment, we use COMM-S and PLANAR-S and set $n = 3$ so as to visualise automaton trajectories in 3D. The experiment aims to show persistent autoencoding, *conditional* generation of 3D point clouds and graph drawing abilities (Eades, 1984; Tamassia, 2013). We randomly sample t in $[15, 25]$ at each optimisation step. We reach an average and *persistent* F1 score of 0.98 and 0.96 for COMM-S and PLANAR-S respectively over 10 different runs. Figure 3 shows the learned dynamics of our autoencoder.

Autoencoding Results E(n)-GNCA autoencoders can scale to higher Euclidean spaces and significantly larger graphs, without neither increasing the size of the models nor losing persistency. This time, we set the coordinate dimension to 8 for all datasets except SBM where it is set to 24, and randomly sample t in $[25, 35]$. Satorras et al. (2021b) already showed the autoencoding superiority of EGNNs compared to classical GNN variants and therefore we only compare against layered EGNNs as a suitable baseline. For a fair comparison, we do *not* allow underlying fully connected graphs for EGNNs, as opposed to Satorras et al. (2021b). Table 1 reports autoencoding results for E(n)-GNCAs and 4-layered EGNNs having 4-times more parameters. Remarkably, E(n)-GNCA outperform layered EGNNs. Examples of graph reconstructions are available in Figure B.5. Furthermore, E(n)-GNCA autoencoders exhibit persistent dynamics (e.g. Figure B.6) for all datasets except SBM, which, given the variable clustered topology, represents the most challenging dataset.

E(n)-GNCAs are multi-target E(n)-GNCA autoencoders are multi-target as they can reach many target states, contrary to what is shown in our previous task and previous work (Grattarola et al., 2021; Mordvintsev et al., 2020; 2022). We suppose this to be the consequence of a more relaxed training objective (cf. Equation 15) than the previous one (cf. Equation 13). In graph autoencoding, in fact, target states are *not* explicitly given but rather a *condition* that they must satisfy is (cf. Equations 14 and 15). Therefore, since we are only interested in reconstructing the ground-truth A via Equation 14, E(n)-GNCAs can converge to any possible configuration from which decoding is possible.

4.3. Simulation of E(n)-equivariant dynamical system

We here show the applicability of E(n)-GNCAs as simulators of E(n)-equivariant dynamical systems. The goal is to learn the transition rule underlying observed trajectories.

Table 1. Autoencoding results averaged over 10 different runs for 4-layered EGNNs and E(n)-GNCAs evaluated at time step $t = 100$.

	E(n)-GNCA		EGNN_4	
	F1↑	BCE↓	F1↑	BCE↓
COMM-S	1.00±0.00	0.05±0.01	0.91±0.03	0.44±0.11
PLANAR-S	0.99±0.01	0.19±0.07	0.87±0.01	0.44±0.05
PLANAR-L	0.98±0.01	0.07±0.03	0.77±0.35	0.34±0.03
PROTEINS	0.95±0.04	0.03±0.02	0.84±0.01	0.08±0.02
SBM	0.92±0.02	0.20±0.02	0.76±0.01	0.34±0.02

Specifically, we train E(n)-GNCAs to simulate the Boids Algorithm (Reynolds, 1987), a 1-step Markovian and distributed multi-agent system designed to simulate flocks of birds using a set of hand-crafted rules. The underlying graph $\mathcal{G}$ is obtained as a fixed-radius nearest neighbourhood of the nodes at each time step, i.e. $\mathcal{G}$ changes dynamically through time. We emphasise that such dynamical system (i) can be formulated as a GCA (cf. Equation 1) and (ii) is E(n)-equivariant. We report more details and show the results of the same experiment for the N-Body problem in Appendix C.

Dataset We extend the 2D simulation of Grattarola et al. (2021) to a 3D space. We create a dataset of 500 trajectories using the ground-truth simulator. Each trajectory has a duration of 500 time steps and is obtained by evolving 100 boids initialised with random positions and velocities.

Training We use attention weights (cf. Equation 8) and Equations 9 and 10 to explicitly account for velocities. We create a mini-batch of randomly sampled sub-trajectories of length $L = 20$. Then, for each mini-batch sub-trajectory $[\mathbf{X}^{(\ell)}, \mathbf{V}^{(\ell)}]_{\ell=1}^{L}$ we input τ_θ with state $\mathbf{S}^{(1)} = [\mathbf{X}^{(1)}, \mathbf{V}^{(1)}, \mathbf{H}^{(1)}]$ and run it for $L-1$ steps obtaining predicted states $[\mathbf{X}'^{(\ell)}, \mathbf{V}'^{(\ell)}, \mathbf{H}'^{(\ell)}]_{\ell=2}^{L}$. Finally, we optimize the MSE of the estimated velocities with the ground truth ones as follows $\mathcal{L}_{\mathsf{MSE}} = \sum_{\ell=2}^{L} \|\mathbf{V}^{(\ell)} - \mathbf{V}'^{(\ell)}\|^2$. Similarly to Satorras et al. (2021b), node features $\mathbf{H}^{(1)}$ are initialised as the output of a linear layer taking $\|\mathbf{V}^{(1)}\|$ as input.

Results As already pointed out by Grattarola et al. (2021), one key aspect of simulating continuous (and chaotic) dynamical systems with GNCAs is that small errors in prediction will quickly accumulate, making it almost impossible for the model to perfectly simulate the true dynamics. Therefore, despite reaching a small validation error, the model *cannot* perfectly approximate the true trajectories. However, following Grattarola et al. (2021), we can quantitatively evaluate the quality of the learned transition rule by using the sample entropy (SE) (Richman & Moorman, 2000) and correlation dimension (CD) (Grassberger & Procaccia, 1983), two measures of complexity for real-valued time series. On average, ground-truth trajectories (of length 500) report an average SE and CD of 0.04 ± 0.01 and 1.02 ± 0.22 respectively, whereas E(n)-GNCA trajectories report 0.04 ± 0.02 and 1.08 ± 0.15 for the same measures. The closeness of the measures indicates that E(n)-GNCA trajectories generate an amount of information comparable to the ground-truth ones, therefore capturing the essence of the underlying rule. Figure 4 shows examples of ground-truth and predicted trajectories.

5. Discussion

We introduced E(n)-GNCAs, isotropic automata showing and promising a wide range of applicability. E(n)-GNCA local interactions have been proven powerful enough to reach globally consistent target conditions (cf. subsections 4.1 and 4.2) and capture complex dynamics (cf. subsection 4.3). To the best of our knowledge, this is the first work proposing isotropic-by-design neural cellular automata.

Limitations The recurrent training of E(n)-GNCAs makes training hard as we faced problems like exploding/vanishing gradients that were mitigated using weight decay and gradient clipping. Further, the 1-to-1 correspondence between initial nodes and target nodes is a strong and non-natural design choice as it leads to abrupt dynamics especially in the first time steps. In future work, we aim to drop this correspondence by adopting Optimal-Transport (Peyré & Cuturi, 2019; Alvarez-Melis et al., 2019).

Broader Impact The possible implications of our work are evident when considering that distributed and self-organizing systems are ubiquitous both in nature (e.g. collective motion, swarming) and technology (e.g. a cyber-physical system operating locally). Furthermore, isometries are very common in dynamical systems (e.g. swarming (Reynolds, 1987), particle simulations (Kipf et al., 2018)) and in many practical applications (e.g. point cloud processing, 3D molecular structures (Ramakrishnan et al., 2014)). Our framework and its inductive biases are particularly useful in all these scenarios, since they allow to learn and discover—rather than hand-design—the transition rules underlying these systems, while accounting for symmetries.

Notably, one of the most remarkable demonstrations of self-organisation can be found in swarm robotics and active matter modeling (Brambilla et al., 2013; Vicsek et al., 1995). Nowadays, we can program tiny robots to locally interact and form a given (E(n)-invariant) pattern, as demonstrated by work such as Mergeable Nervous Systems (Mathews et al., 2017) and Kilobots (Rubenstein et al., 2012). To the best of our knowledge, such programs are currently designed by humans. We hope our work could encourage a line of research in which GNNs further unlock the power of GCAs to implement a desired behavior through differentiable, distributed and emergent computation.

References

Adamatzky, A. *Game of life cellular automata*, volume 1. Springer, 2010. doi:10.1007/978-1-84996-217-9.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

Alvarez-Melis, D., Jegelka, S., and Jaakkola, T. S. Towards optimal transport with global invariances. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1870–1879. PMLR, 2019.

Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013. doi:10.1007/s11721-012-0075-2.

Chan, B. W.-C. Lenia: Biology of artificial life. *Complex Systems*, 28(3):251–286, 2019. doi:10.25088/ComplexSystems.28.3.251.

Defferrard, M., Martin, L., Pena, R., and Perraudin, N. PyGSP: Graph signal processing in Python, 2017. URL https://github.com/epfl-lts2/pygsp/.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003. doi:10.1016/S0022-2836(03)00628-4.

Eades, P. A heuristic for graph drawing. *Congressus numerantium*, 42:149–160, 1984.

Elmenreich, W. and Fehérvári, I. Evolving self-organizing cellular automata based on neural network genotypes. In *Self-Organizing Systems*, volume 6557 of *Lecture Notes in Computer Science*, pp. 16–25. Springer, 2011. doi:10.1007/978-3-642-19167-1_2.

Falcon, W. and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL https://github.com/Lightning-AI/lightning. Version 1.4.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. doi:10.48550/arXiv.1903.02428.

Foramitti, J. AgentPy: A package for agent-based modeling in Python. *Journal of Open Source Software*, 6(62):3065, 2021. doi:10.21105/joss.03065.

Gilpin, W. Cellular automata as convolutional neural networks. *Physcal Review E*, 100:032402, Sep 2019. doi:10.1103/PhysRevE.100.032402.

Godwin, J., Schaarschmidt, M., Gaunt, A. L., Sanchez-Gonzalez, A., Rubanova, Y., Veličković, P., Kirkpatrick, J., and Battaglia, P. Simple GNN regularisation for 3D molecular property prediction and beyond. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=1wVvweK3oIb.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, volume 2, pp. 729–734, 2005. doi:10.1109/IJCNN.2005.1555942.

Grassberger, P. and Procaccia, I. Characterization of strange attractors. *Physical Review Letters*, 50(5):346–349, Jan 1983. doi:10.1103/PhysRevLett.50.346.

Grattarola, D., Livi, L., and Alippi, C. Learning graph cellular automata. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20983–20994, 2021. URL https://proceedings.neurips.cc/paper/2021/file/af87f7cdcda223c41c3f3ef05a3aaeea-Paper.pdf.

Ha, D. and Tang, Y. Collective intelligence for deep learning: A survey of recent developments. *Collective Intelligence*, 1(1):26339137221114874, 2022. doi:10.1177/26339137221114874.

Huang, W., Mordatch, I., and Pathak, D. One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4455–4464. PMLR, 2020. URL https://proceedings.mlr.press/v119/huang20d.html.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. doi:10.48550/arXiv.1412.6980.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2688–2697. PMLR, 2018. URL https://proceedings.mlr.press/v80/kipf18a.html.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. In *Bayesian Deep Learning Workshop (NIPS 2016)*, 2016. doi:10.48550/arXiv.1611.07308.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*,

2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 3538–3545. AAAI Press, 2018. doi:10.48550/arXiv.1801.07606.

Lillicrap, T. P. and Santoro, A. Backpropagation through time and the brain. *Current Opinion in Neurobiology*, 55: 82–89, 2019. doi:10.1016/j.conb.2019.01.011.

Lipton, Z. C., Berkowitz, J., and Elkan, C. A critical review of recurrent neural networks for sequence learning, 2015. URL https://arxiv.org/abs/1506.00019.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/file/1e44fdf9c44d7328fecc02d677ed704d-Paper.pdf.

Martinkus, K., Loukas, A., Perraudin, N., and Wattenhofer, R. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. *arXiv preprint arXiv:2204.01613*, 2022.

Mathews, N., Christensen, A. L., O'Grady, R., Mondada, F., and Dorigo, M. Mergeable nervous systems for robots. *Nature communications*, 8(1):1–7, 2017.

Mitchell, M. *Complexity: A guided tour*. Oxford university press, 2009.

Mocz, P. Create your own n-body simulation (with python), September 2020. URL https://medium.com/swlh/create-your-own-n-body-simulation-with-python-f417234885e9. [Online; posted 14-September-2020].

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. Growing neural cellular automata. *Distill*, 5(2):e23, 2020. doi:10.23915/distill.00023.

Mordvintsev, A., Randazzo, E., and Fouts, C. Growing isotropic neural cellular automata. *arXiv preprint arXiv:2205.01681*, 2022.

Nichele, S., Ose, M. B., Risi, S., and Tufte, G. Ca-neat: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

Palm, R. B., Duque, M. G., Sudhakaran, S., and Risi, S. Variational neural cellular automata. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=7fFO4cMBx_9.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peyré, G. and Cuturi, M. Computational optimal transport: With applications to data science. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019. doi:10.1561/2200000073. URL https://arxiv.org/abs/1803.00567.

Pérez, J., Marinković, J., and Barceló, P. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyGBdo0qFm.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Randazzo, E., Mordvintsev, A., Niklasson, E., Levin, M., and Greydanus, S. Self-classifying mnist digits. *Distill*, 5(8):e00027–002, 2020.

Rendell, P. Turing universality of the game of life. In *Collision-based computing*, pp. 513–539. Springer, 2002.

Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25–34, 1987.

Richman, J. S. and Moorman, J. R. Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*, 278(6):H2039–H2049, 2000.

Rubenstein, M., Ahler, C., and Nagpal, R. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE international conference on robotics and automation*, pp. 3293–3298. IEEE, 2012.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 333–341. SIAM, 2021.

Satorras, V. G., Hoogeboom, E., Fuchs, F., Posner, I., and Welling, M. E(n) equivariant normalizing flows. *Advances in Neural Information Processing Systems*, 34: 4181–4192, 2021a.

Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) equivariant graph neural networks. In *International conference on machine learning*, pp. 9323–9332. PMLR, 2021b.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Sudhakaran, S., Grbic, D., Li, S., Katona, A., Najarro, E., Glanois, C., and Risi, S. Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737*, 2021.

Tamassia, R. *Handbook of graph drawing and visualization*. CRC press, 2013.

Tesfaldet, M., Nowrouzezahrai, D., and Pal, C. Attention-based neural cellular automata. *arXiv preprint arXiv:2211.01233*, 2022.

Turing, A. M. The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1):153–197, 1990.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vicsek, T., Czirók, A., Ben-Jacob, E., Cohen, I., and Shochet, O. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226, 1995.

von Neumann, J. The general and logical theory of automata. In Taub, A. H. (ed.), *Collected works — Design of computers, theory of automata and numerical analysis*, volume V, pp. 288–328. Pergamom Press, 1963.

Wenkel, F., Min, Y., Hirn, M., Perlmutter, M., and Wolf, G. Overcoming oversmoothness in graph convolutional networks via hybrid scattering networks. *arXiv preprint arXiv:2201.08932*, 2022.

Wolfram, S. *Cellular automata and complexity: collected papers*. crc Press, 2018.

Wulff, N. and Hertz, J. A. Learning cellular automaton dynamics with neural networks. *Advances in Neural Information Processing Systems*, 5:631–638, 1992.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2019.

Zhou, K., Dong, Y., Wang, K., Lee, W. S., Hooi, B., Xu, H., and Feng, J. Understanding and resolving performance degradation in deep graph convolutional networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2728–2737, 2021.

A. Pattern Formation

A.1. Implementations details

Model Let n, h and m be coordinate, hidden state and message dimension respectively. We set $h = 16$ and $m = 32$, and $n = 2$ or $n = 3$ depending on the target geometric graph. Our MLPs (5,300 parameters overall) are then defined as follows:

- Message MLP $\phi_m : \mathbb{R}^{2h+1} \to \mathbb{R}^m$ (cf. Equation 3): $\big[\mathsf{LinearLayer}(2h+1, m), \mathsf{TanH}(), \mathsf{LinearLayer}(m, m), \mathsf{TanH}()\big]$,
- Coordinate MLP $\phi_x : \mathbb{R}^m \to \mathbb{R}^1$ (cf. Equation 4): $\big[\mathsf{LinearLayer}(m, m), \mathsf{TanH}(), \mathsf{LinearLayer}(m, 1), \mathsf{TanH}()\big]$,
- Hidden state MLP $\phi_h : \mathbb{R}^{h+m} \to \mathbb{R}^h$ (cf. Equation 7): $\big[\mathsf{LinearLayer}(m+h, h), \mathsf{TanH}(), \mathsf{LinearLayer}(h, h)\big]$.

We normalise node features $\mathbf{H}$ using PairNorm (Zhao & Akoglu, 2019) after every transition rule step.

Training We train the model by minimising Equation 13, using Adam (Kingma & Ba, 2015) with initial learning rate of 0.0005. The learning rate is then decreased during training using a reduce-on-plateau schedule. We use gradient clipping and weight decay as regularisation techniques. We increase the batch size during training (from a minimum of 4 to 32) so as to promote a faster convergence, and that is because a small batch size leads to more frequent re-initialisation of the pool states in the early training steps. We train the model to convergence, by monitoring the validation loss with a fixed patience of training steps. For all details, we refer the reader to our code.

A.2. Full trajectories

In this subsection, we report the full trajectories of our model for several geometric graphs considered.

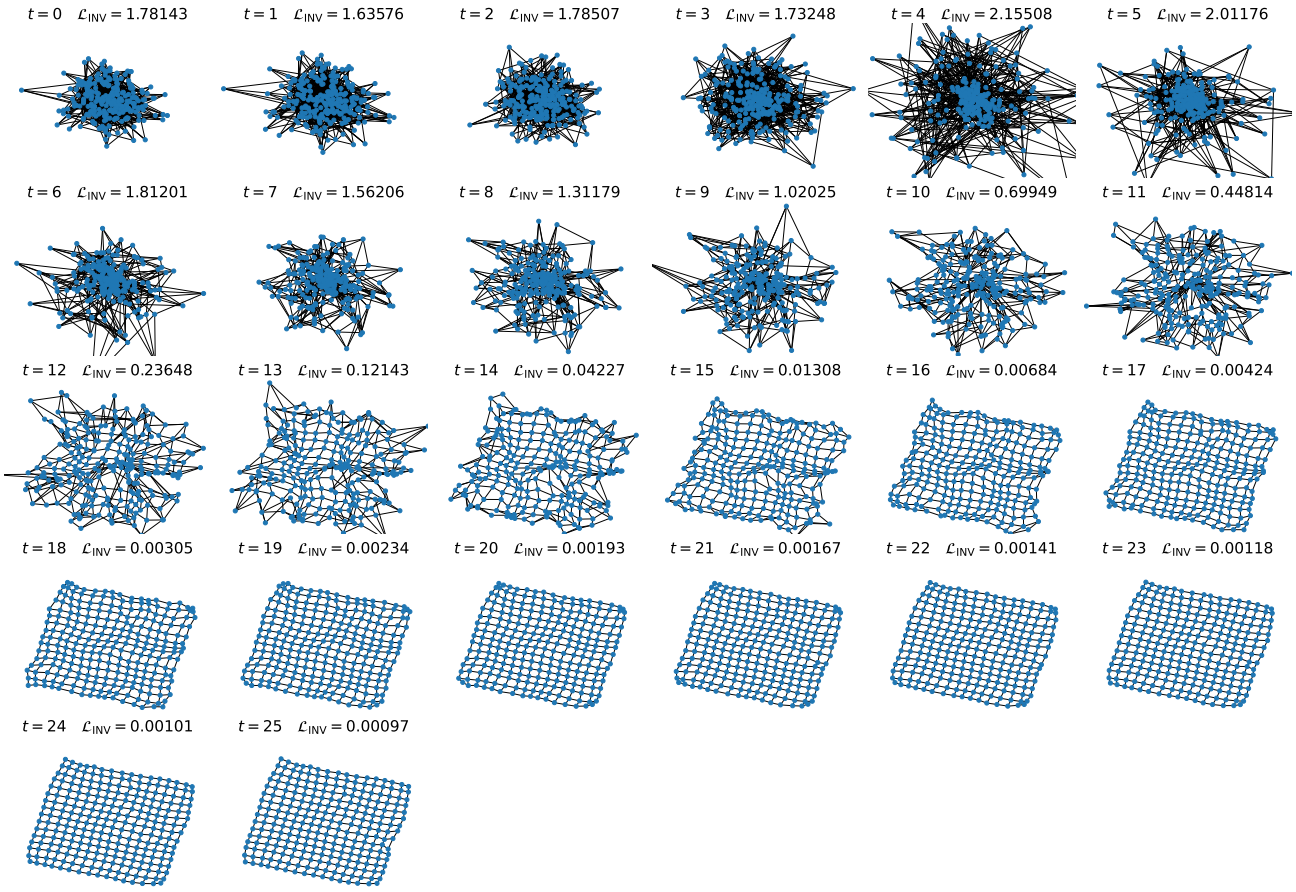

Figure A.1. Convergence to a 2D grid. We report the loss value (cf. Equation 13) in each figure. Best viewed digitally and zoomed in.

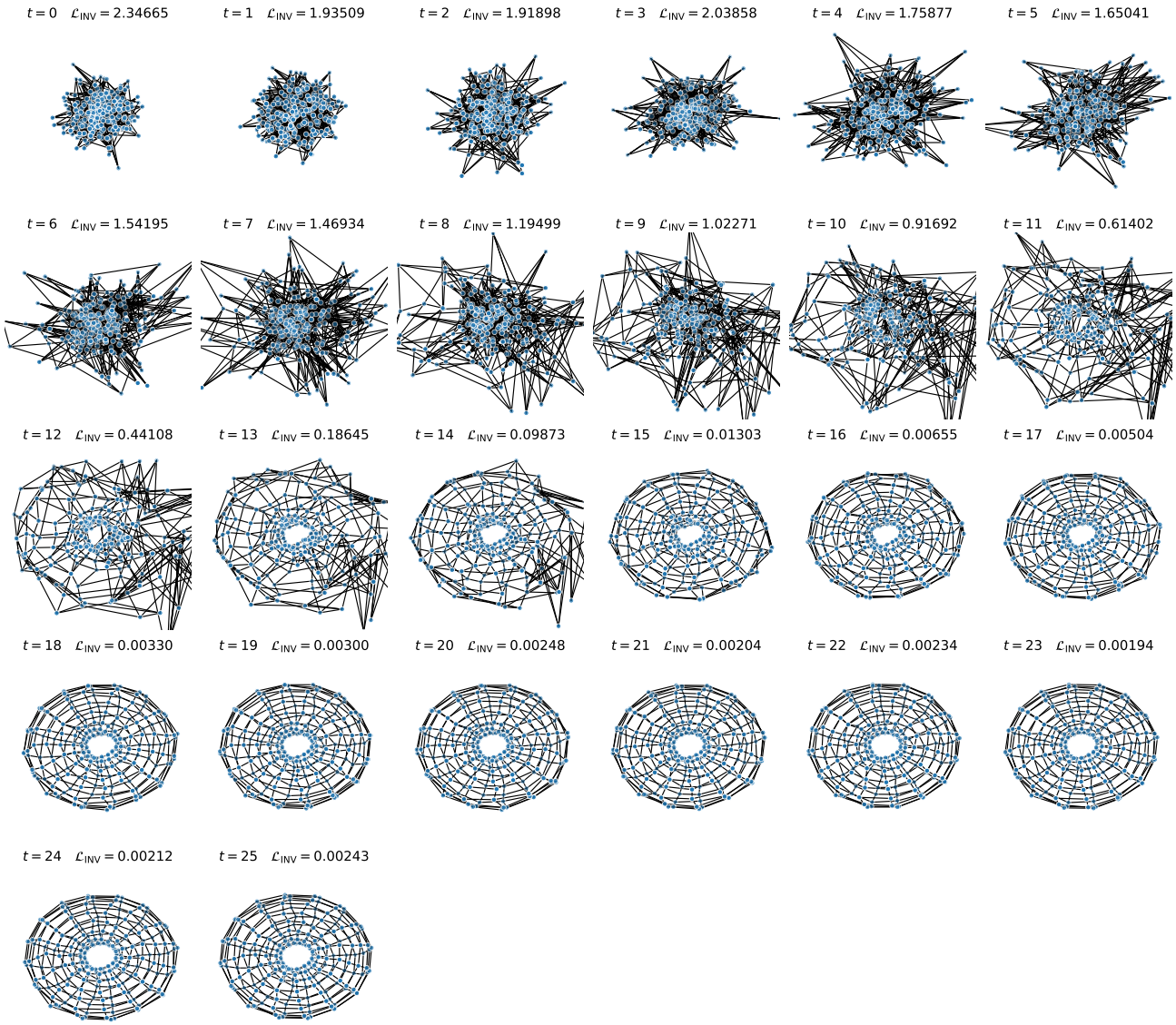

Figure A.2. Convergence to a 3D torus. We report the loss value (cf. Equation 13) in each figure. Best viewed digitally and zoomed in.

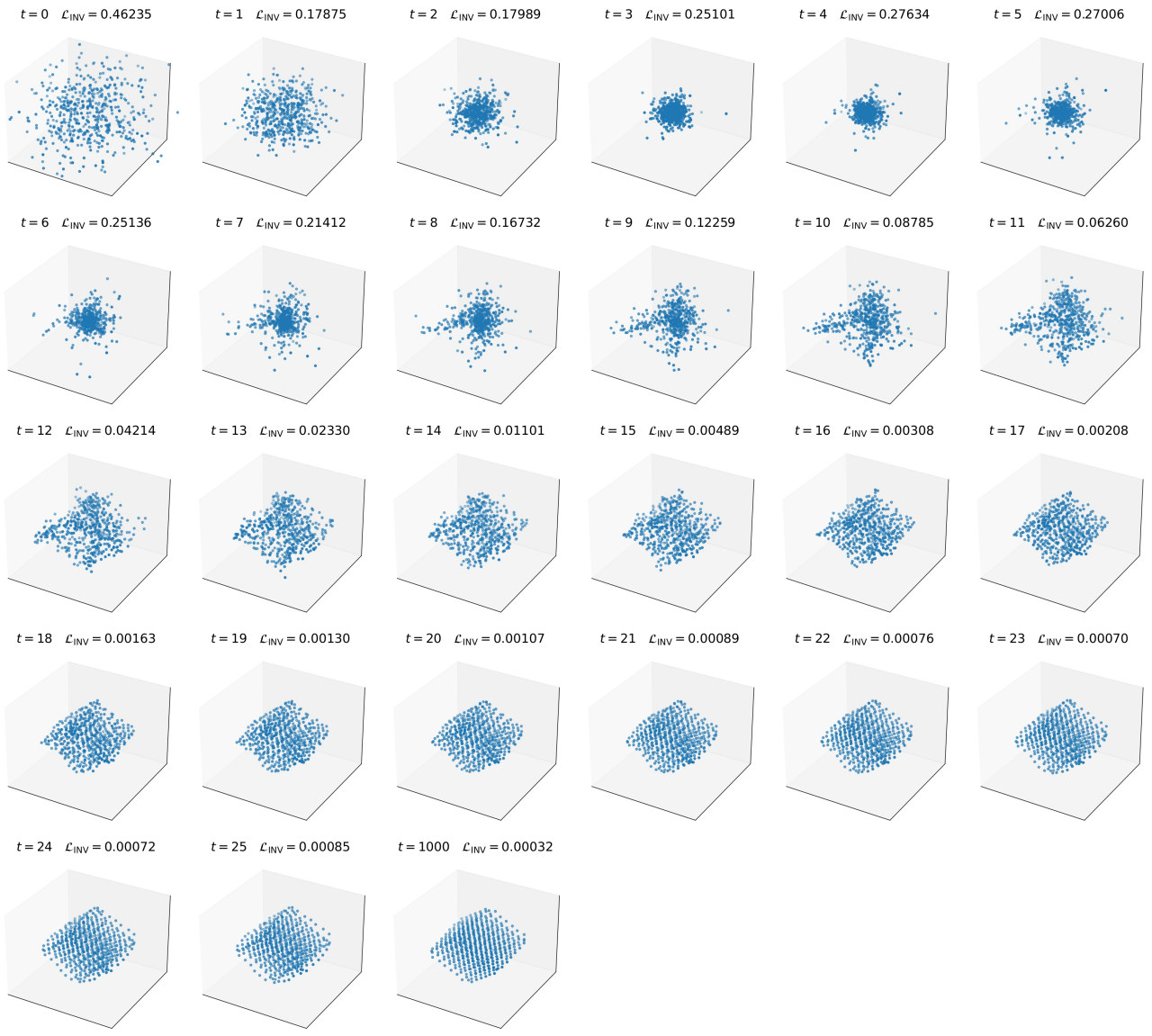

Figure A.3. Convergence to a 3D Cube. We report the loss value (cf. Equation 13) in each figure. To avoid clutter, nearest-neighbor edges are not shown. Best viewed digitally and zoomed in.

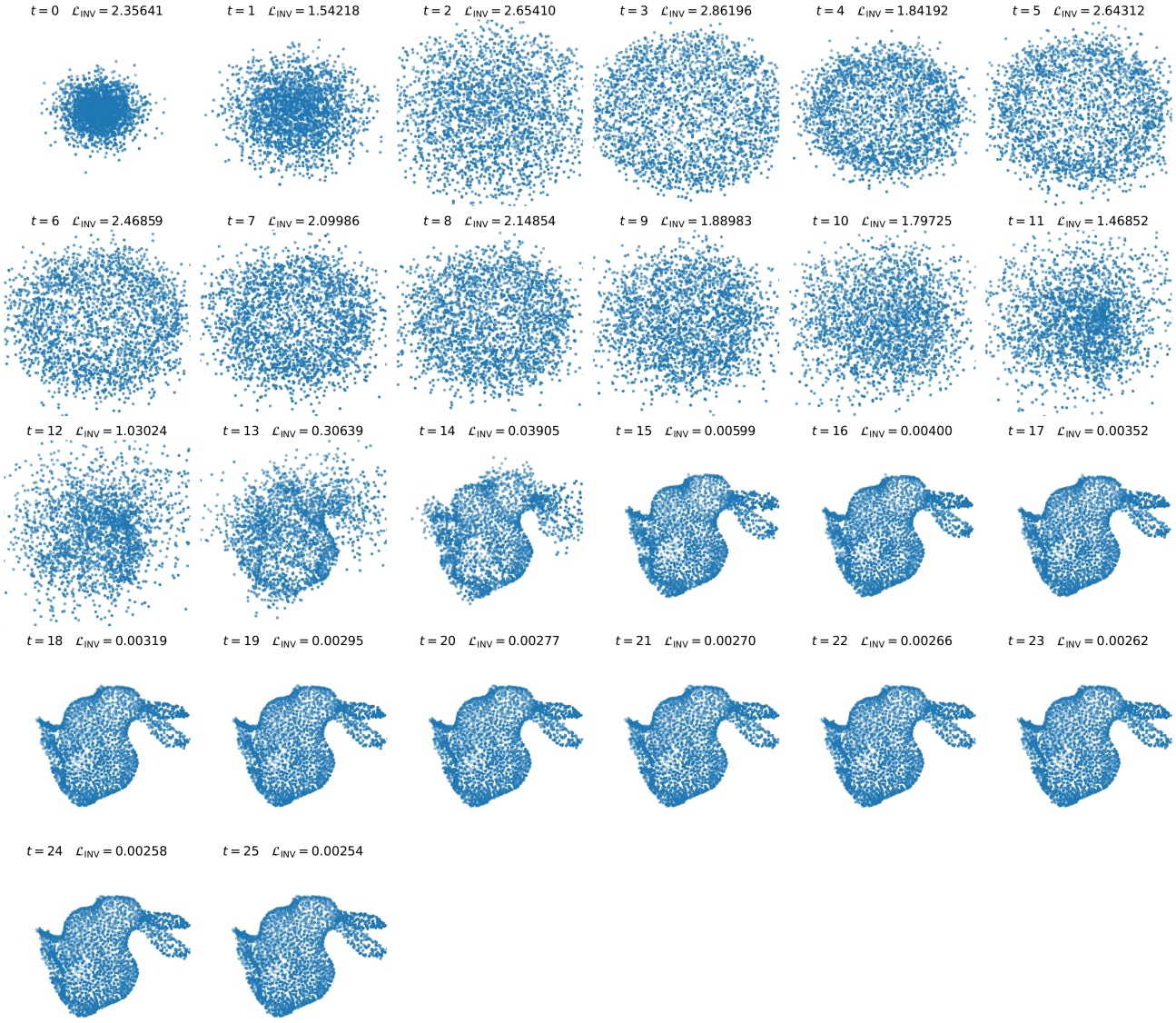

Figure A.4. Convergence to the Stanford bunny. We report the loss value (cf. Equation 13) in each figure. To avoid clutter, nearest-neighbor edges are not shown. Best viewed digitally and zoomed in.

B. Graph autoencoding with Cellular Automata

B.1. Implementation details

Model We use the exact same architecture as the one detailed in Appendix A.

Training We train the model to minimise Equation 14, using Adam (Kingma & Ba, 2015) with initial learning rate of 0.0005. The learning rate is then decreased during training using a reduce-on-plateau schedule. We set the batch size to 32 and train the model by monitoring the validation loss with a patience of 20 epochs. We use negative edge sampling when computing the loss so as to have a balanced supervisory signal when processing sparse graphs. For all details, we refer the reader to our code.

Testing In order to effectively evaluate the quality of a graph reconstruction, we first need to binarise its soft adjacency matrix $\hat{A} \in [0,1]^{|\mathcal{V}|\times|\mathcal{V}|}$ (cf. Equation 14). Therefore, at test time, we fine-tune a threshold $\hat{t} \in (0,1)$ on the validation set as to maximize the F1 score of validation-set graph reconstructions. Once we have threshold $\hat{t}$, we can binarise soft adjecency matrices of test-set graphs and then compute the F1 scores thereof.

B.2. Reconstructions

Figure B.5 shows examples of test-set reconstructions from our autoencoding task (cf. subsection 4.2, Table 1).

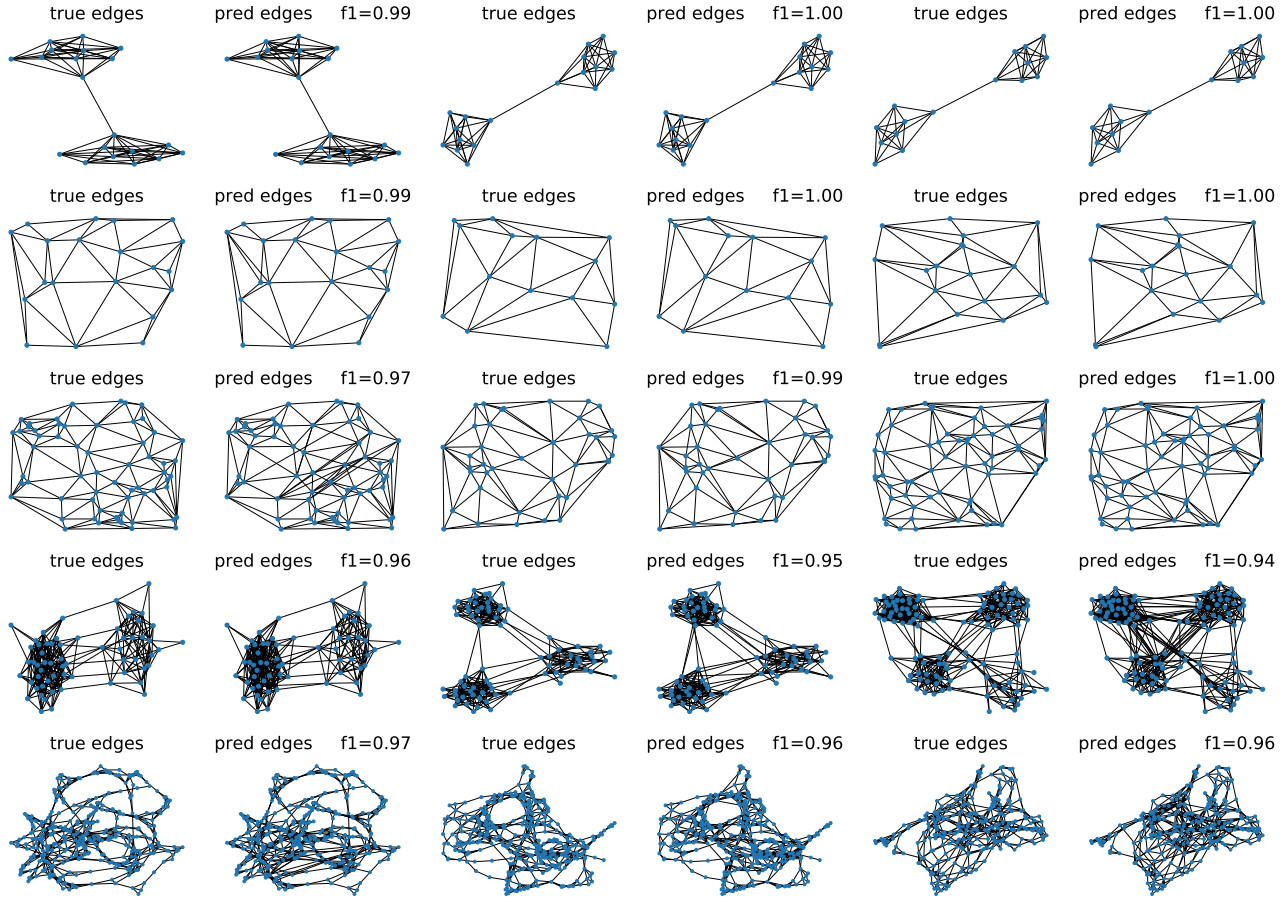

Figure B.5. Test-set graph reconstructions at time step $t = 100$ for COMM-S (1st row), PLANAR-S (2nd row), PLANAR-L (3rd row), PROTEINS (4th row) and SBM (5th row). In alternating columns, we first have ground truth graphs and then respective reconstructions with F1 scores attached on top. The position of the nodes is fixed for both ground-truth and reconstructed graphs so as to visually inspect the quality of the reconstructions. Best viewed digitally and zoomed in.

B.3. Persistency test

Figure B.6 shows the F1-score trend over time for E(*n*)-GNCA autoencoders (cf. subsection 4.2, Table 1).

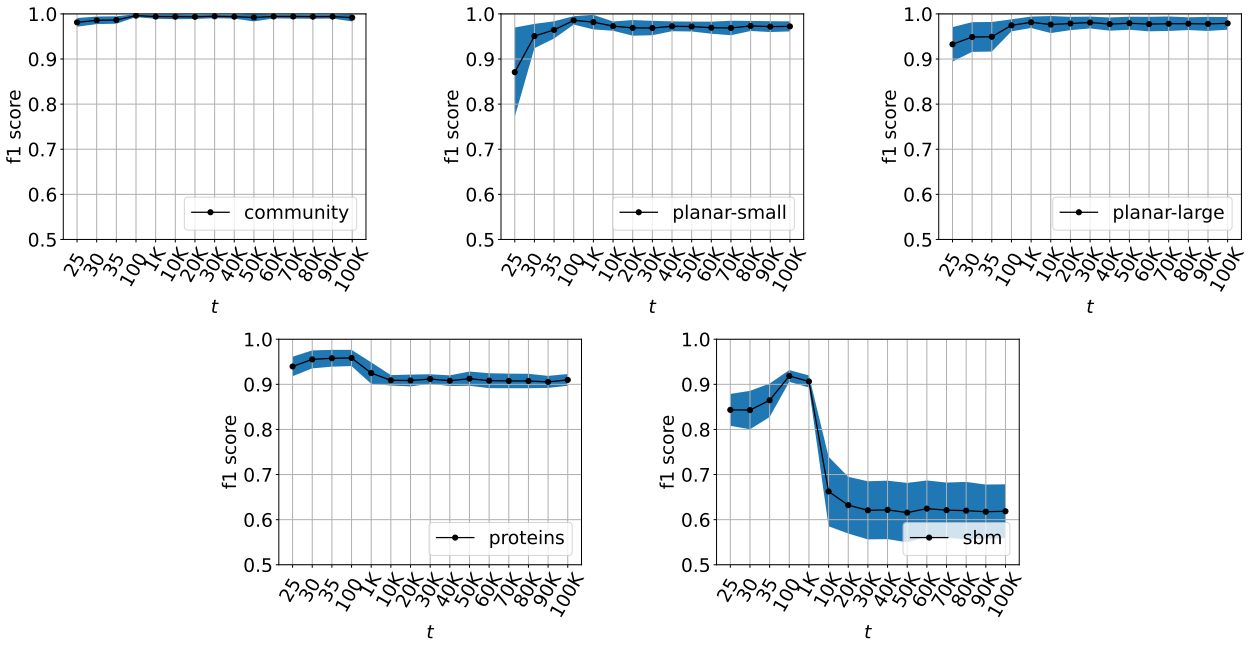

Figure B.6. Persistency test for E(*n*)-GNCA autoencoders. We run transition rule τ_θ for up to 100,000 time steps and report the F1-score (y-axis) at different time steps (x-axis) for each dataset. Trends are averaged over 10 different runs. E(*n*)-GNCAs exhibit a persistent autoencoding trend for all datasets except SBM, which, given its clustered topology, represents the most challenging dataset.

C. Simulation of E(*n*)-equivariant dynamical system

C.1. Implementation Details

Model Let n, h and m be coordinate, hidden state and message dimension respectively. We set $h = 16$, $m = 32$ and $n = 3$. Our MLPs (5500 parameters overall) are then defined as follows:

- Message MLP $\phi_m : \mathbb{R}^{2h+1} \to \mathbb{R}^m$ (cf. Equation 3): $\big[\mathsf{LinearLayer}(2h+1, m), \mathsf{TanH}(), \mathsf{LinearLayer}(m, m), \mathsf{TanH}()\big]$,
- Attention MLP $\phi_a : \mathbb{R}^m \to [0, 1]^1$ (cf. Equation 8): $\big[\mathsf{LinearLayer}(m, 1), \mathsf{Sigmoid}()\big]$,
- Velocity MLP $\phi_v : \mathbb{R}^{h+1} \to \mathbb{R}^1$ (cf. Equation 9): $\big[\mathsf{LinearLayer}(m, m), \mathsf{TanH}(), \mathsf{LinearLayer}(m, 1), \mathsf{TanH}()\big]$,
- Coordinate MLP $\phi_x : \mathbb{R}^m \to \mathbb{R}^1$ (cf. Equation 9): $\big[\mathsf{LinearLayer}(h+1, h/2), \mathsf{TanH}(), \mathsf{LinearLayer}(h/2, 1)\big]$,
- Hidden state MLP $\phi_h : \mathbb{R}^{h+m} \to \mathbb{R}^h$ (cf. Equation 7): $\big[\mathsf{LinearLayer}(m+h, h), \mathsf{TanH}(), \mathsf{LinearLayer}(h, h)\big]$.

Training We train the model to minimise the MSE between groud-truth and predicted trajectories, using Adam (Kingma & Ba, 2015) with initial learning rate of 0.001. The learning rate is then decreased during training using a reduce-on-plateau schedule. We set the batch size to 16 and train the model by monitoring the validation loss with a patience of 20 epochs. For all details, we refer the reader to our code.

C.2. The Boids Algorithm

Our implementation of the Boids algorithm is mainly inspired by the one available in AgentPy (Foramitti, 2021), which in turn is based on the seminal work of Reynolds (1987). Each boid has location $\mathbf{x} \in \mathbb{R}^3$ and velocity $\mathbf{v} \in \mathbb{R}^3$. The simulation takes place in a squared 3D fix-sized box. We compute the underlying graph $\mathcal{G}$ at each step of the algorithm by connecting

the boids that are within a given radius from each other. At each step of the algorithm, we sequentially apply the following transformations synchronously to all boids to compute the change in each boid velocity and location:

- A *cohesion* force is applied to bring the position of each boid closer to its neighbours;
- An *alignment* force is applied to match the velocity of each boid to the average velocity of its neighbours;
- If the distance between a boid and its neighbours is lower than a user-defined threshold, a *separation* force is applied to steer the boid away from these excessively close neighbours;
- If a boid is within a user-defined radius from the bounding box, a force is applied to steer the boid towards the centre;
- For each boid, the resulting force is summed to the current velocity thus determining the new velocity;
- Finally, the position of each boid is updated according to the new velocity.

For all details, we refer the reader to our code.

C.3. N-body Problem

Our implementation of the N-body dynamical system is taken from Mocz (2020). Similarly to Satorras et al. (2021b), we create a dataset of 5000 trajectories. Each trajectory has a duration of 1000 time steps and is obtained by evolving 5 bodies initialised with random positions and velocities. Importantly, the N-body problem assumes an underlying fully-connected graph $\mathcal{G}$. The experiment aims to investigate a use case in which E(n)-GNCA nodes are allowed to globally communicate. For all details, we refer the reader to our code.

Results We quantitatively evaluate the quality of the learned transition rule using two complexity measures: sample entropy (SE) (Richman & Moorman, 2000) and correlation dimension (CD) (Grassberger & Procaccia, 1983). The ground-truth trajectories report an average SE and CD of 0.07 ± 0.05 and 1.42 ± 0.49 respectively, whereas E(n)-GNCA trajectories report 0.07 ± 0.04 and 1.46 ± 0.40 for the same measures. The closeness of the measures indicates that E(n)-GNCA trajectories generate an amount of information comparable to the ground-truth ones, therefore capturing the underlying rule. Figure C.7 shows examples of ground-truth and predicted trajectories.

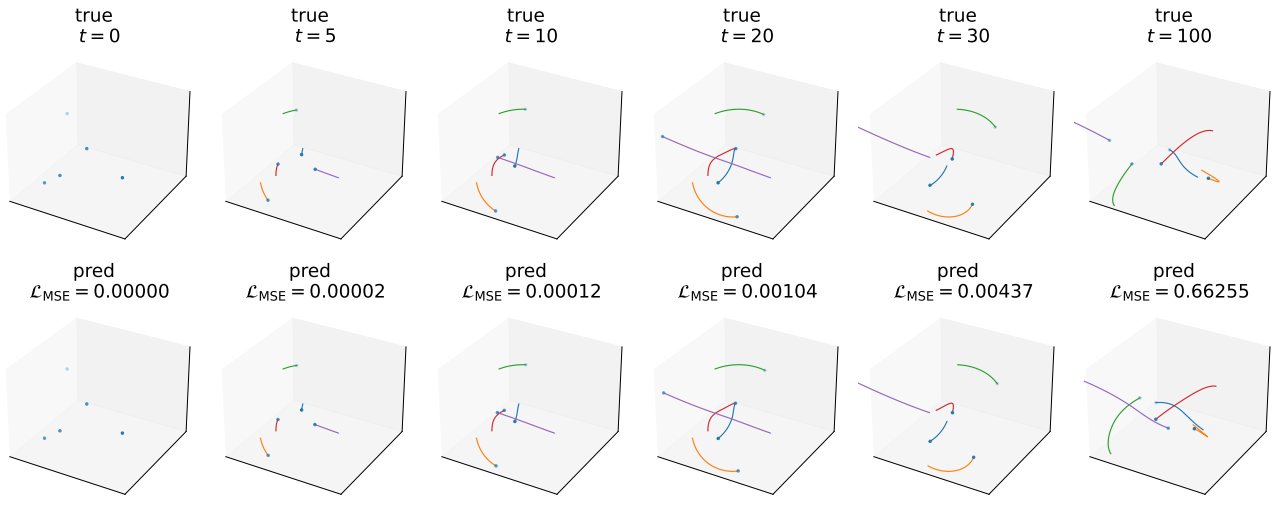

Figure C.7. N-Body problem simulation. First row shows a ground-truth trajectory at different time steps. Second row shows E(n)-GNCA predicted trajectory and reports the MSE w.r.t. the true one at each time step.