arTs Università degli Studi di Trieste
Archivio della ricerca – postprint

# Generative abstraction of Markov population processes ☆

Francesca Cairoli *, Fabio Anselmi, Alberto d'Onofrio, Luca Bortolussi

*Department of Mathematics and Geoscience, University of Trieste, Trieste, Italy*

A B S T R A C T

Markov population models are a widespread formalism used to model the dynamics of complex systems, with applications in systems biology and many other fields. The associated Markov stochastic process in continuous time is often analyzed by simulation, which can be costly for large or stiff systems, particularly when a massive number of simulations has to be performed, e.g. in a multi-scale model. A strategy to reduce computational load is to abstract the population model, replacing it with a simpler stochastic model, faster to simulate. Here we pursue this idea, exploring and comparing state-of-the-art generative models, which are flexible enough to automatically learn distributions over entire trajectories, rather than single simulation steps, from observed realizations of the system. In particular, we compare a Generative Adversarial setting with a Score-based Diffusion approach and show how the latter outperforms the former both in terms of accuracy and stability at the cost of slightly higher simulation times. To improve the accuracy of abstract samples, we develop an active learning framework to enrich our dataset with observations whose expected satisfaction of a temporal requirement differs significantly from the abstract one. We experimentally show how the proposed abstractions are well suited to work on multi-scale and data-driven scenarios, meaning that we can infer a (black-box) dynamical model from a pool of real data.

## 1. Introduction

A wide range of complex systems can be modelled as a network of chemical reactions (CRN). Stochastic simulation avoids the explicit construction of the state space and thus it is typically the only feasible analysis approach that scales in a computationally tractable manner with the increase in system size. The well-known Gillespie Stochastic Simulation Algorithm (SSA) [1] is widely used for simulating models, as it samples from the exact distribution over trajectories. This algorithm is effective to simulate systems of moderate complexity, but it does not scale well to systems with many species and reactions, large populations, or internal stiffness. In these scenarios, a more effective choice is to rely on approximate simulation algorithms such as tau-leaping [2] and hybrid simulation [3]. Nonetheless, when the number of simulations required is extremely large and possibly costly, e.g. when one needs to simulate a large population of heterogeneous cells

☆ This article belongs to Section C: Theory of natural computing, Edited by Lila Kari.

* Corresponding author.
*E-mail address:* francesca.cairoli@units.it (F. Cairoli).

in a multi-scale model of a tissue or to simulate many heterogeneous individuals in a population ecology scenario, all these methods become extremely computationally demanding, even for HPC facilities.

A viable approach to address such a problem is *model abstraction*, which aims at reducing the underlying complexity of the model, and thus reduces its simulation cost. However, building effective model abstractions is difficult, requiring a lot of ingenuity and manpower. Here we advocate the strategy of *automatically* learn an abstraction from simulation data. Our strategy is to frame model abstraction as a conditional unsupervised learning problem and learn an abstract probabilistic model using state-of-the-art deep learning techniques. The probabilistic model should be able to generate approximate trajectories efficiently and in constant time, i.e., independent of the complexity of the original system, thus sensibly reducing the simulation cost.

*Related work.* The idea of using machine learning as a model abstraction tool to approximate and simplify the dynamics of a Markov Population Process has received some attention in recent years. In [4] the authors use a Mixture Density Network (MDN) [5] to approximate the transition kernel of the stochastic process. In [6] the authors extend the previous approach by introducing an automated search of the MDN architecture that better fits the data. In [7] the authors present a Bayesian model abstraction technique, based on Dirichlet Processes, that allows the quantification of the reconstruction uncertainty. In all cases, what is learned is an approximate transition kernel, i.e., the probabilistic distribution of a single simulation step. In [8] the authors propose a reinforcement learning framework to infer some statistics of the high-dimensional chemical master equations using Kolmogorov's backward equation. All the aforementioned approaches focus on learning the distribution of the state of the system after a time $\Delta t$, the so-called transition kernel. However, such approaches perform poorly when the time interval is small and the dynamics is transient, showing a clear propagation of the error as the approximate kernel is applied iteratively to form a trajectory.

Instead of learning an approximate transition kernel, in [9], the authors focused on learning one-dimensional marginal distributions of the CRN dynamics as a function of time. The authors themselves acknowledge the inability to capture complex dynamics. In contrast, our approach is capable of learning the joint distribution over the entire state space, preserving the correlation among species over time. Furthermore, our experimental evaluation demonstrates how our method effectively captures complex dynamics. In [10], a Mixture Density Network (MDN) is used to approximate the Fisher Information of a CRN at discrete time points under partial observability. While these results can be used to perform fast policy search, they do not explicitly reconstruct the dynamics but only a functional of the latter. Recently, several approaches [11–13] have been proposed to accelerate the simulation of Markovian and non-Markovian models via machine learning approximation. These methods however, assume full knowledge of the dynamics at hand, making them excellent to improve scalability but leaving no room for a data-driven approach. As a matter of fact, if the assumed CRN structure is incorrect, predictions become unreliable. In contrast, our approach requires no structural assumptions over the CRN that generates the data, allows for partial observability and, thus, can be used in a purely data-driven fashion. In [14] the authors learn the distribution of an entire trajectory of fixed length. This latter problem is not solvable with any of the previously adopted techniques, and its major goal is to keep abstraction error under control. In fact, training the abstract model on a full trajectory, rather than on pairs of subsequent states, allows the abstract model to retain and capture more information about the dynamics of the Markov process. Furthermore, producing a full trajectory reduces, even more, the computational cost of simulating a large pool of trajectories for different initial settings. Conditional Wasserstein Generative Adversarial Nets (cWGAN), a strong and flexible technique to learn probabilistic models, were used in [14]. The GAN-based model abstraction technique is capable of learning a conditional distribution over the trajectory space, keeping into account the correlation, both spatial and temporal, among all the different species and conditioning both on initial states and model parameters. Despite the good results presented in [14], we feel like, in light of recent advances in deep generative modelling, those results could be considerably improved, both in terms of reconstruction accuracy and stability of the solution.

*Contributions.* This paper extends [14] with the following contributions:

- We compare the performances of the cWGAN approach with those of a conditional score-based diffusion model (CSDI), a novel state-of-the-art approach to learn probabilistic models over time series.
- We show how CSDI outperforms cWGAN w.r.t. accuracy and stability of the solution and how it is slightly slower than cWGAN in generating new samples.
- We introduce an active learning framework to stir the generative models towards a better reconstruction of the level of satisfaction of a signal temporal logic (STL) requirement.
- We test the performances of both techniques on a multi-scale case study using SSA as the baseline.

*Paper structure.* The paper is organized as follows: Section 2 introduces the relevant background notions. Section 3 describes the generative abstraction procedure in detail. Section 4 presents the case studies, Section 5 the experimental settings and Section 6 the numerical results. Conclusions are drawn in Section 7.

## 2. Background

### 2.1. Chemical reaction networks

Consider a system with $n$ species $\mathsf{X}_1, \ldots, \mathsf{X}_n$ evolving according to a stochastic model defined as a Chemical Reaction Network (CRN). The state $s_{i,t}$, $i = 1, \ldots, n$, denote the number of individuals of species $\mathsf{X}_i$ present in the population at time $t$. The dynamics of a CRN is described by a set of reactions $\mathsf{R}_1, \ldots, \mathsf{R}_m$. The firing of reaction $\mathsf{R}_j$ results in a transition of the system from state $s_t = (s_{1,t}, \ldots, s_{n,t}) \in S \subseteq \mathbb{N}^n$ to state $s_t + \nu_j$, with $\nu_j$ being the update vector. Under the well-stirred assumption, the time evolution can be modelled as a Continuous Time Markov Chain (CTMC) on a discrete state space. Due to the memoryless property of CTMCs, the probability of finding the system in state $s$ at time $t$ given that it was in state $s_0$ at time $t_0$ can be expressed as a system of ODEs known as Chemical Master Equation (CME):

$$\partial_t p_{s_0}(s_t = s) = \sum_{j=1}^{m} \left[ f_{\theta_j}^{\mathsf{R}_j}(s - \nu_j) p_{s_0}(s_t = s - \nu_j) - f_{\theta_j}^{\mathsf{R}_j}(s) p_{s_0}(s_t = s) \right], \tag{1}$$

for every $s_0, s \in S$, where $S \subseteq \mathbb{N}^n$ denotes the state space and $\Theta \subseteq \mathbb{R}^m$ denotes the parameter space for all the propensity functions $f_{\theta_j}^{\mathsf{R}_j}$. Since in general the CME is a system with countably many differential equations, its analytic or numeric solution is almost always unfeasible. An alternative computational approach is to generate trajectories using stochastic algorithms for simulation, like the well-known Gillespie's SSA [1] which produces statistically correct trajectories, i.e., sampled according to the stochastic process described by the CME in (1).

### 2.2. Signal temporal logic

Signal temporal logic (STL) [15,16] formulas provide an expressive language to describe system behaviours so that they are easy to interpret. System requirements can be expressed via Signal Temporal Logic (STL), which enables the specification of properties of dense-time, real-valued signals $\vec{s} : \mathbb{T} \to \mathbb{R}$, where $\vec{s}(t) = s_t$, and the automatic generation of monitors for testing properties on individual trajectories. The rationale of STL is to transform real-valued signals into Boolean ones, using formulae built on the following *STL syntax*: $\phi := true \mid \mu \mid \neg\phi \mid \phi \wedge \phi \mid \phi \; U_I \phi$, where $I \subseteq \mathbb{T}$ is a temporal interval, either bounded, $I = [a, b]$, or unbounded, $I = [a, +\infty)$, for any $0 \leq a < b$. Atomic propositions $\mu$ are (non-linear) inequalities on the states of a signal $\vec{s}$ at a time $t$, $\mu = (g(\vec{s}(t)) > 0)$, where $g : S \to \mathbb{R}$ and $\vec{s}(t)$ is a state in $S$. From this essential syntax, it is easy to define other operators, used to abbreviate the syntax in an STL formula: $false := \neg true$, $\phi \vee \psi := \neg(\neg\phi \wedge \neg\psi)$, $F_I := true \; U_I \phi$ and $G_I := \neg F_I \neg\phi$. Monitoring the satisfaction of a formula is done recursively by leveraging the tree structure of the STL formula. The satisfaction relation is defined as follows.

$$\begin{aligned}
(\vec{s}, t) \models \mu \quad &\Leftrightarrow \quad g(\vec{s}(t)) > 0 \\
(\vec{s}, t) \models \neg\phi \quad &\Leftrightarrow \quad \neg((\vec{s}, t) \models \phi) \\
(\vec{s}, t) \models \phi_1 \wedge \phi_2 \quad &\Leftrightarrow \quad (\vec{s}, t) \models \phi_1 \wedge (\vec{s}, t) \models \phi_2 \\
(\vec{s}, t) \models \phi_1 U_{a,b} \phi_2 \quad &\Leftrightarrow \quad \exists\, t' \in [t + a, t + b] \text{ s.t. } (\vec{s}, t') \models \phi_2 \wedge \\
&\qquad\qquad \forall\, t'' \in [t, t'],\ (\vec{s}, t'') \models \phi_1
\end{aligned}$$

*Quantitative semantics.* The robustness of a trajectory quantifies the level of satisfaction w.r.t. $\phi$. Positive robustness means that the property is satisfied, whereas negative robustness means that the property is violated. Robustness is denoted as a function $\rho_\phi : S^H \times T \to \mathbb{R}$ that maps a given signal $\vec{s}$ of length $H$, a formula $\phi$ and a time $t$ to some real value, $\rho_\phi(\vec{s}, t) \in \mathbb{R}$. It measures the maximum perturbation that can be applied to the signal without changing its truth value w.r.t. $\phi$.

Similarly to the Boolean semantics, the quantitative semantics of a formula $\phi$ over a signal $\vec{s}$ is defined recursively over the tree structure of the STL formula, as described below:

$$\begin{aligned}
\rho_\mu(\vec{s}, t) \quad &= g(\vec{s}(t)) \\
\rho_{\neg\phi}(\vec{s}, t) \quad &= -\rho_\phi(\vec{s}, t) \\
\rho_{\phi_1 \wedge \phi_2}(\vec{s}, t) \quad &= \min(\rho_{\phi_1}(\vec{s}, t), \rho_{\phi_2}(\vec{s}, t)) \\
\rho_{\phi_1 U_{[a,b]} \phi_2}(\vec{s}, t) \quad &= \sup_{t' \in [t+a, t+b]} \Big( \min\big(\rho_{\phi_2}(\vec{s}, t'), \inf_{t'' \in [t, t']} \rho_{\phi_1}(\vec{s}, t'')\big)\Big).
\end{aligned}$$

The sign of $\rho_\phi$ indicates the Boolean satisfaction of signal $\vec{s}$. In particular, $\rho_\phi(\vec{s}, t) > 0 \Rightarrow (\vec{s}, t) \models \phi$ and $\rho_\phi(\vec{s}, t) < 0 \Rightarrow (\vec{s}, t) \not\models \phi$.

### 2.3. Deep generative models

Every dataset can be considered as a set of observations $\mathbf{x}$ drawn from an unknown distribution $p(\mathbf{x})$. Generative models aim at learning a model that mimics this unknown distribution as closely as possible, i.e., learn a parametric distribution $p_\psi(\mathbf{x})$ as similar as possible to $p(\mathbf{x})$. The parametric distribution $p_\psi$ is chosen so that one knows how to efficiently sample from it. These samples are new, meaning not previously observed, but look as if they could have belonged to the original dataset. The rationale of generative modelling, in general, is to generate data from noise. A generative model acts as a distribution transformer, i.e., a map $G_\psi : Z \to X$ transforming a simple distribution $p_Z$ over a latent space $Z$ into a complex distribution over the state space $X$. Given a noise sample $\mathbf{z} \sim p_Z(\mathbf{z})$, $G_\psi(\mathbf{z})$ is thus a sample in $X$. Similarly, conditional generative models can be seen as maps $G_\psi : Z \times Y \to X$, where $Y$ is the conditioning space. We can generate samples in $X$ conditioned on $\mathbf{y} \in Y$, $G_\psi(\mathbf{z}, \mathbf{y}) \in X$, approximating the distribution $p(\mathbf{x} \mid \mathbf{y})$.

#### 2.3.1. Generative Adversarial Nets

Generative Adversarial Nets (GANs) [17] are a class of deep learning-based generative models, that, given a dataset, are capable of generating new random but plausible examples.

*Wasserstein GAN.* In this work, as in [14], we consider the Wasserstein version of GAN (WGAN) [18,19] as it is known to be more stable and less sensitive to the choice of model architecture and hyperparameters compared to a traditional GAN. WGANs use the Wasserstein distance (also known as Earth-Mover's distance), rather than the Jensen Shannon divergence, to measure the difference between the model distribution $p_\psi$ and the target distribution $p$. Because of Kantorovich-Rubinstein duality [20] such distance can be computed as the supremum over all the 1-Lipschitz functions $f : X \to \mathbb{R}$:

$$W(p, p_\psi) = \sup_{||f||_L \leq 1} \left( \mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_\psi}[f(\mathbf{x})] \right). \tag{2}$$

We approximate these functions $f$ with a neural net $C_{w_c}$ parametrized by its weights $w_c$. To enforce the Lipschitz constraint we follow [19] and introduce a penalty over the norm of the gradients. It is known that a differentiable function is 1-Lipchitz if and only if it has gradients with norm at most 1 everywhere. The objective function, to be maximized w.r.t. $w_c$, becomes:

$$\mathcal{L}(w_c, \psi) := \mathbb{E}_{\mathbf{x} \sim p}[C_{w_c}(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_\psi}[C_{w_c}(\mathbf{x})] - \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}}(\|\nabla_{\hat{\mathbf{x}}} C_{w_c}(\hat{\mathbf{x}})\|_2 - 1)^2], \tag{3}$$

where $\lambda$ is the penalty coefficient and $p_{\hat{\mathbf{x}}}$ is defined by sampling uniformly along straight lines between pairs of points sampled from $p$ and $p_\psi$. This is actually a softer constraint that however performs well in practice [19]. The $C_{w_c}$ network is referred to as *critic* and it outputs different scores for real and fake samples, its objective function (Eq. (3)) provides an estimate of the Wasserstein distance among the two distributions. On the other hand, the distribution $p_\psi$ is parametrized by $\psi$; we seek the parameters $\psi$ that make it as close as possible to $p$. To achieve this, we consider a random variable $Z$ with a fixed simple distribution $p_Z$ and pass it through a parametric function, the *generator*, $G_\psi : Z \to X$ that generates samples following the distribution $p_\psi$. Therefore, the WGAN architecture consists of two deep neural nets, a generator that proposes a distribution and a critic that estimates the distance between the proposed and the real (unknown) distribution. Using WGAN brings several important advantages compared to traditional GAN: it avoids the mode collapse problem, which makes WGAN more suitable for capturing stochastic dynamics, it drastically reduces the problem of vanishing gradients and it also has an objective function that correlates with the quality of generated samples, making the results easier to interpret.

*Conditional GAN.* Conditional Generative Adversarial Nets (cGAN) [21] are a type of GANs that involves the conditional generation of examples, i.e., the generator produces examples of a required type, e.g. examples that belong to a certain class, and thus they introduce control over the desired generated output. In our application, we want the generation of stochastic trajectories to be conditioned on some model parameters $\theta$ and on the initial state of the system $s_0$, i.e. $\mathbf{y} = (\theta, s_0) \in Y := S \times \Theta$.

Furthermore, dealing with inputs that are trajectories, i.e. sequences of fixed length, requires the use of convolutional neural networks (CNNs) [22] for both the generator and the critic. The architecture used in this work is thus a conditional Wasserstein Convolutional GAN with gradient penalty, it is going to be referred to as cWGAN-GP.

#### 2.3.2. Score-based diffusion models

Let $\mathbf{x}^0$ denote samples coming from the unknown data distribution $\mathbf{x}^0 \sim p(\mathbf{x}^0)$ over a space $X$ and let $\mathbf{x}^\tau$ for $\tau = 1, \ldots, T$ be a sequence of variables in the same space $X$ of $\mathbf{x}^0$. Let $p_\psi(\mathbf{x}^0)$ be a distribution that approximates $p(\mathbf{x}^0)$. Diffusion probabilistic models are latent variable models composed of two processes: a forward and a reverse process.

The *forward process* is a Markov process designed to iteratively turn a sample of $p(\mathbf{x}^0)$ into pure noise (typically a standard Gaussian distribution). It is defined by the following Markov chain:

$$p(\mathbf{x}^{1:T} | \mathbf{x}^0) = \prod_{\tau=1}^{T} p(\mathbf{x}^\tau | \mathbf{x}^{\tau-1}) \tag{4}$$

where $p(\mathbf{x}^\tau|\mathbf{x}^{\tau-1}) := \mathcal{N}\left(\sqrt{1-\beta_\tau}\mathbf{x}^{\tau-1}, \beta_\tau \mathbb{I}\right)$ and $\beta_\tau$ is a positive constant representing the noise level introduced at diffusion step $\tau$. The conditional distribution $p(\mathbf{x}^\tau|\mathbf{x}^0)$ can be written in closed form as $p(\mathbf{x}^\tau|\mathbf{x}^0) = \mathcal{N}\left(\sqrt{\alpha_\tau}\mathbf{x}^0, (1-\alpha_\tau)\mathbb{I}\right)$, where $\alpha_\tau = \prod_{i=0}^{\tau}(1-\beta_i)$. Thus, $\mathbf{x}^\tau$ can be expressed as $\mathbf{x}^\tau = \sqrt{\alpha_\tau}\mathbf{x}^0 + (1-\alpha_\tau)\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

The *reverse process* inverts back in time the forward process, denoising $\mathbf{x}^\tau$ to recover $\mathbf{x}^0$. It is defined by the following reverse Markov chain:

$$\begin{aligned} p_\psi(\mathbf{x}^{0:T}) &:= p_\psi(\mathbf{x}^T) \prod_{\tau=1}^{T} p_\psi(\mathbf{x}^{\tau-1}|\mathbf{x}^\tau), \quad \mathbf{x}^T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ p_\psi(\mathbf{x}^{\tau-1}|\mathbf{x}^\tau) &:= \mathcal{N}\left(\mathbf{x}^{\tau-1}; \mu_\psi(\mathbf{x}^\tau, \tau), \sigma_\psi(\mathbf{x}^\tau, \tau)\mathbf{I}\right). \end{aligned} \tag{5}$$

In [23] the following specific parameterization of $p_\psi(\mathbf{x}^{\tau-1}|\mathbf{x}^\tau)$ has been proposed:

$$\begin{aligned} \mu_\psi(\mathbf{x}^\tau, \tau) &= \frac{1}{\alpha_\tau}\left(\mathbf{x}^\tau - \frac{\beta_\tau}{\sqrt{1-\alpha_\tau}}\epsilon_\psi(\mathbf{x}^\tau, \tau)\right) \\ \sigma_\psi^2(\mathbf{x}^\tau, \tau) &= \begin{cases} \frac{1-\alpha_{\tau-1}}{1-\alpha_\tau}\beta_\tau, & \tau > 1 \\ \beta_1, & \tau = 1 \end{cases} \end{aligned} \tag{6}$$

where $\epsilon_\psi : X \times \mathbb{R} \to X$ is a trainable denoising function. The reverse process is not analytically computable and it has to be trained from data by solving the following optimization problem:

$$\min_\psi \mathbb{E}_{\mathbf{x}^0 \sim p(\mathbf{x}^0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tau} ||\epsilon - \epsilon_\psi(\mathbf{x}^\tau, \tau)||_2^2. \tag{7}$$

The denoising function $\epsilon_\psi$ estimates the noise vector $\epsilon$ that was added to its noisy input $\mathbf{x}^\tau$. This training objective can also be viewed as a weighted combination of denoising score matching used for training score-based generative models [24–26]. Once trained, we can sample $\mathbf{x}^0$ from (5). So the sample generating function $G_\psi$ can be defined as the concatenation of sampling procedures along the $T$ diffusion steps. The sampling can be formalized as $\bar{\mathbf{x}}^T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\bar{\mathbf{x}}^{T-1} \sim p_\psi(\mathbf{x}^{T-1}|\bar{\mathbf{x}}^T)$ until $\bar{\mathbf{x}}^0 \sim p_\psi(\mathbf{x}^0|\bar{\mathbf{x}}^1)$ which is our desired sample in $X$.

*Conditional score-based diffusion.* Given a sample $\mathbf{x}^0$ whose observation is conditional on a variable $\mathbf{y}^0$, we need to consider a conditional diffusion model that estimates the probability $p(\mathbf{x}^0|\mathbf{y}^0)$. The reverse process aims at modelling $p(\mathbf{x}^{\tau-1}|\mathbf{x}^\tau, \mathbf{y}^0)$. The denoising function becomes $\epsilon_\psi : \left(X \times \mathbb{R} \mid Y\right) \to X$ and the optimization becomes

$$\min_\psi \mathbb{E}_{(\mathbf{x}^0, \mathbf{y}^0) \sim p(\mathbf{x}^0, \mathbf{y}^0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tau} ||\epsilon - \epsilon_\psi(\mathbf{x}^\tau, \tau|\mathbf{y}^0)||_2^2. \tag{8}$$

## 3. Model abstraction

Let's now formulate the problem of automatically learning an abstraction of a Markov stochastic process from a pool of simulated data. The underlying idea is the following: given a stochastic process $\{s_t\}_{t\geq 0}$ with transition probabilities $\mathbb{P}_{s_0}(s_t = s) = \mathbb{P}(s_t = s \mid s_{t_0} = s_0)$, we aim at finding another stochastic process whose trajectories are faster to simulate but similar to the original ones. Time has to be discretized, meaning we fix an initial time $t_0$ and a time step $\Delta t$ that suits our problem. We define $\tilde{s}_i := s_{t_0 + i\cdot\Delta t}$, $\forall i \in \mathbb{N}$. In addition, given a fixed time horizon $H$, we define time-bounded trajectories as $\xi_{[1,H]} = \tilde{s}_1\tilde{s}_2\cdots\tilde{s}_H \in S^H \subseteq \mathbb{N}^{H\times n}$. Given a state $s_0$ and a set of parameters $\theta$, we can represent a trajectory of length $H$ as a realization of a random variable over the state space $S^H$. The probability distribution for such random variable is given by the product of the transition probabilities at each time step:

$$\mathbb{P}_{s_0,\theta}(\xi_{[1,H]} = \tilde{s}_1\tilde{s}_2\cdots\tilde{s}_H) = \prod_{i=1}^{H} \mathbb{P}_{\tilde{s}_{i-1},\theta}(\xi_{[1,H]}(t_i) = \tilde{s}_i).$$

The CTMC, $\{s_t\}_{t\geq 0}$, is now expressed as a time-homogeneous Discrete Time Markov Chain $\{\tilde{s}_i\}_i$. An additional approximation has to be made: the abstract model takes values in $S' \subseteq \mathbb{R}^n_{\geq 0}$, a continuous space in which the state space $S \subseteq \mathbb{N}^n$ is embedded. In constructing the approximate probability distribution for trajectories we can decide to restrict our attention to arbitrary aspects of the process, rather than trying to preserve the full behaviour. A *projection* $\pi$ from $S^H$ to an arbitrary space $U^H$ can be used to reach this purpose, for instance, to monitor the number of molecules belonging to a certain subset of chemical species, i.e., $U \subseteq S$. Note that $\pi(\xi_{[0,H]})$ is a random variable over $U^H$. Such flexibility could be extremely helpful in capturing the dynamics of systems in which some species are not observable.

*Abstraction accuracy.* Another important ingredient is a meaningful quantification of the error introduced by the abstraction procedure, i.e., the reconstruction accuracy. Such quantification must be based on a distance, $d$, among distributions. We choose the Wasserstein distance. Given a distribution over initial states $s_0$ and a distribution over parameters $\theta$, we would like to measure the expected error at every time instant $t_i = t_0 + i \cdot \Delta t$ with $i \in \{1, \dots, H\}$. Formally, we want to measure $\mathbb{E}_{s_0,\theta}\left[d\big(\pi(\xi_{[1,H]})\big|_i, \pi'(\hat{\xi}_{[1,H]})\big|_i\big)\right]$ where $\pi(\xi_{[1,H]})\big|_i$ denotes the $i$-th time components of the projected trajectory $\pi(\xi_{[1,H]}) \in U^H$ and $\hat{\xi}_{[1,H]}$ denotes, in general, an abstract trajectory. To estimate such quantity we use a well-known unbiased estimator, which is the average over the distances computed over a large sample set of initial settings. Computing the distance among SSA and abstract distributions at each time step quantifies how small the expected error is and, more importantly, how it evolves in time. As a matter of fact, it shows whether the error tends to propagate or not and how much each species contributes to the abstraction error. In practice, we compute $H \cdot n$ distances among distributions over $\mathbb{N}$ as we want to know how each species contributes in the reconstruction error.

### 3.1. Dataset generation

*Training set.* Choose a set of $N_{train}$ initial settings and for each setting simulate $M_{train}$ SSA trajectory of length $H$. The training set is composed of $N_{train} \cdot M_{train}$ pairs initial setting-trajectory, i.e., pairs $(\mathbf{y}_i, \xi^{ij}_{[1,H]})$ for $i = 1, \dots, N_{train}$ and $j = 1, \dots, M_{train}$ where $\mathbf{y}_i = (\theta^i, s_0^i)$:

$$\mathcal{D}_{train} = \left\{ \left( (\theta^h, s_0^h), \xi^h_{[1,H]} \right) \right\}_{h=1}^{N_{train} \times M_{train}}.$$

*Test set.* Choose a set of $N_{test}$ initial settings and for each setting simulate a large number, $M_{test} \gg M_{train}$, of SSA trajectory of length $H$. The test set is composed of $N_{test} \cdot M_{test}$ pairs initial setting-trajectory, i.e., pairs $(\mathbf{y}_i, \xi^{ij}_{[1,H]})$ for $i = 1, \dots, N_{test}$ and $j = 1, \dots, M_{test}$:

$$\mathcal{D}_{test} = \left\{ \left( (\theta^h, s_0^h), \xi^h_{[1,H]} \right) \right\}_{h=1}^{N_{test} \times M_{test}}.$$

*Partial observability.* In case of partial observability, $U \subseteq S$, we fix an initial condition for species in $U$, and simulate a pool of trajectories each time sampling the initial value of species in $S \setminus U$. As a result, we are learning an abstract distribution that marginalizes over unobserved variables.

### 3.2. Property-based active learning

Deep generative models are highly efficient and accurate tools to abstract a stochastic process. However, they are black-box models trained to minimize a specific measure of error and receive no reward for preserving a certain property of the system, for instance, biological consistency properties. As previously stated, deep generative models can be seen as deterministic functions $G_\psi$ mapping the initial setting $\mathbf{y} = (\theta, s_0)$ and a latent sample of noise $\mathbf{z}$ into abstract trajectories $\hat{\xi}_{[1,H]}$. Mathematically, $G_\psi(\mathbf{y}, \mathbf{z}) = \hat{\xi}_{[1,H]}$.

In this work, we propose an active learning framework with a query strategy based on the expected ability of the abstract model to reproduce a certain STL property $\phi$, more precisely, based on how well it reconstructs its robustness. The rationale is the following: given an STL property $\phi$ and a validation set $\mathcal{D}_v$, composed of $N_v$ initial settings, $\mathbf{y}_1, \dots, \mathbf{y}_{N_v}$, with $M_v$ trajectories per each setting, we compute the robustness of each trajectory in $\mathcal{D}_v$ w.r.t. $\phi$. We then take the average robustness w.r.t. each initial condition, i.e., the average over the $M_v$ trajectories starting at that specific point. Given an initial condition $\mathbf{y}_i$, for $i \in \{1, \dots, N_v\}$, we define the *residual error in robustness estimates*, $\delta_i$, as the absolute distance between the average robustness over the $M_v$ observed trajectories $\bar{\rho}_\phi(\mathbf{y}_i) := \frac{1}{M_v} \sum_{j=0}^{M_v} \rho_\phi(\xi^{ij}_{[1,H]})$ and the average robustness over the $M_v$ trajectories generated by the abstract model $\bar{\rho}^\psi_\phi(\mathbf{y}_i) := \frac{1}{M_v} \sum_{j=0}^{M_v} \rho_\phi(G_\psi(\mathbf{y}, \mathbf{z}_j))$, where $\mathbf{z}_1, \dots, \mathbf{z}_{M_v} \sim p_Z(\mathbf{z})$. Mathematically,

$$\delta_i = \mid \bar{\rho}_\phi(\mathbf{y}_i) - \bar{\rho}^\psi_\phi(\mathbf{y}_i) \mid . \tag{9}$$

As STL robustness can be easily made differentiable, the quantity above could also be explicitly added to the objective function of the generative models. However, it would considerably slow down the training process and it would make the inferred model property-specific. A new property would require full retraining. Our active learning framework, on the other hand, uses this quantity only to fine-tune a pre-trained model that do not depend on any property. A Gaussian Process Regressor (GPR) is used to infer the distribution of residual errors over the entire input space Y. As long as the dimension of the validation set, i.e., the number of initial conditions, is relatively small, exact analytic GP inference can be performed with a cost cubic in the number of initial conditions.[1] Evaluating the fitted GP at a novel condition $\mathbf{y}_* = (\theta^*, s_0^*)$

[1] If the number of initial conditions is too large, we can leverage sparsification methods to approximate GP inference [27,28].

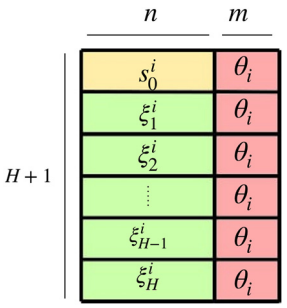


**Fig. 1.** Concatenated input of the critic net $C_{w_c}$.

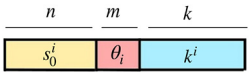


**Fig. 2.** Concatenated input of the generator net $G_\psi$.

results in a Gaussian distribution with mean $\mu_*$ and standard deviation $\sigma_*$. We define the *upper confidence bound* (UCB) as $\text{ucb}(\mathbf{y}_*) := \mu_* + \sigma_*$. After GPR inference, we evaluate the UCB over a pool of $q$ initial conditions, $\text{ucb}(\mathbf{y}_1), \dots, \text{ucb}(\mathbf{y}_q)$, and use only the points with the highest values of the UCB to fine-tune the pre-trained model. In practice, we need to simulate the $M_{train}$ SSA trajectories and feed this new data to the pre-trained generative model that would update its weights accordingly. The decision threshold defining how many points are queried from the pool is chosen as a quantile of the empirical distribution of the error over the pool of size $q$. Finally, the fine-tuning will stir the generative models to better reconstruct the scenarios presenting high gaps in the reconstruction of the average STL robustness. This active learning strategy enhances the data efficiency of the learning process, which is crucial when dealing with complex real-world systems.

The abstract function $G_\psi$ defined above strongly depends on the generative model considered. Below we better describe how the two approaches differ in building $G_\psi$. Moreover, we detail the structure of the proposed framework, where deep generative models are exploited to infer abstractions over the trajectory space.

### 3.3. WGAN-based abstraction

*cWCGAN-GP structure.* The critic $C_{w_c}$ takes as input a batch of initial states, $s_0^1, \dots, s_0^b$, a batch of parameters, $\theta_1, \dots, \theta_b$, and a batch of subsequent trajectories, $\xi_{[1,H]}^1, \dots, \xi_{[1,H]}^b$. For each $i \in \{1, \dots, b\}$ the inputs, $\xi_{[1,H]}^i$, $s_0^i$ and $\theta_i$, are concatenated to form an input with dimension $b \times (H+1) \times (n+m)$ (see Fig. 1). Formally, $C_{w_c} : S^{H+1} \times \Theta \to \mathbb{R}$. To enforce the Lipschitz property over $C_{w_c}$ we add a gradient penalty term over $\mathbb{P}_{\hat{\mathbf{x}}}$. Samples of $\mathbb{P}_{\hat{\mathbf{x}}}$ are generated by sampling uniformly along straight lines connecting points coming from a batch of real trajectories and points coming from a batch of generated trajectories.

On the other hand, the generator $G_\psi$ takes as input a batch of initial states, $s_0^1, \dots, s_0^b$, a batch of parameters, $\theta_1, \dots, \theta_b$, and a batch of random noise, $\mathbf{z}^1, \dots, \mathbf{z}^b$, with dimension $k$, a user-defined hyper-parameter. For each $i \in \{1, \dots, b\}$ the two inputs are, once again, concatenated to form an input with dimension $b \times (n+m+k)$ (see Fig. 2). The generator outputs a batch of generated trajectories $\hat{\xi}_{[1,H]}^1, \dots \hat{\xi}_{[1,H]}^b$. Formally, $G_\psi : S \times \Theta \times Z \to S^H$, such that $G_\psi(s_0, \theta, z) = \hat{\xi}_{[1,H]} = s_1 \cdots s_H$.

*Model training.* The cWCGAN-GP-based model abstraction framework consists in training two different CNNs. The loss function, introduced in Eq. (3), is a parametric function depending both on the generator weights $\psi$ and the critic weights $w_c$. When training the critic, we keep the generator weights constant $\overline{\psi}$, and we maximize $\mathcal{L}(w_c, \overline{\psi})$ w.r.t. $w_c$. Formally, we solve the problem

$$w_c^* = \underset{w_c}{\text{argmax}} \Big\{ \mathcal{L}(w_c, \overline{\psi}) \Big\}.$$

On the other hand, in training the generator, we keep the critic weights constant $\overline{w}_c$, and we minimize $\mathcal{L}(\overline{w}_c, \psi)$ w.r.t. $\psi$. Formally, we solve the problem

$$\psi^* = \underset{\psi}{\text{argmin}} \Big\{ \mathcal{L}(\overline{w}_c, \psi) \Big\} = \underset{\psi}{\text{argmin}} \Big\{ -\mathbb{E}_{z,(s_0,\theta)} \Big[ C_{\overline{w}_c}(G_\psi(\mathbf{z}, s_0, \theta), s_0, \theta) \Big] \Big\}.$$

As mentioned in Section 2, the loss function derives from the Wasserstein distance between the real and generated distributions, see [18,19] for the mathematical details.

Intuitively, the generator generates a batch of samples, and these, along with real examples from the dataset, are provided to the critic, which is then updated to get better at estimating the distance between the real and the abstract distribution

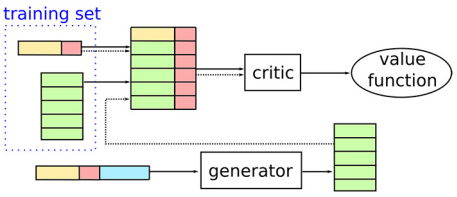


**Fig. 3.** Diagram of the cWCGAN architecture.

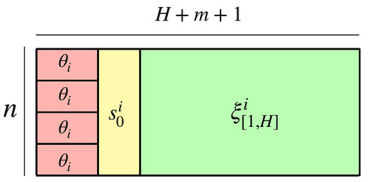


**Fig. 4.** Concatenated input of the CSDI.

(see the diagram in Fig. 3). The generator is then updated based on scores obtained by the generated samples from the critic. An important collateral advantage is that WGANs have a loss function that correlates with the quality of generated examples.

Training the cWCGAN-GP has a cost. Nonetheless, once it has been trained, its evaluation is extremely fast. Details about training and evaluation costs are discussed in Section 5.

*Abstract model simulation.* Once the training is over, we can discard the critic and focus only on the trained generator $G_\psi$. In order to generate an abstract trajectory starting from a state $s_0^*$ with parameters $\theta^*$, we just have to sample a value $\mathbf{z}$ from the random noise distribution $p_Z$ and evaluate the generator on the pair $(s_0^*, \theta^*, \mathbf{z})$. The output is a stochastic trajectory of length $H$: $G_\psi(s_0^*, \theta^*, \mathbf{z}) = \hat{\xi}_{[1,H]}$. The stochasticity is provided by the random noise variable, de facto the generator acts as a distribution transformer that maps a simple random variable into a complex distribution. In order to generate a pool of $M$ trajectories, we simply sample $M$ different values from the random noise variable: $\mathbf{z}_1, \ldots, \mathbf{z}_M$. Therefore, the generation of a trajectory has a fixed computational cost.

### 3.4. Score-based abstraction

*CSDI structure.* As before, we take as input a batch of initial states, $s_0^1, \ldots, s_0^b$, a batch of parameters, $\theta_1, \ldots, \theta_b$, and a batch of subsequent trajectories, $\xi^1_{[1,H]}, \ldots, \xi^b_{[1,H]}$. For each $i \in \{1, \ldots, b\}$, we concatenate the initial condition $(\theta^i, s_0^i)$ with the trajectory $\xi^{ij}_{[1,H]}$ obtaining an input with dimension $b \times n \times (H + m + 1)$ (see Fig. 4). We then define a mask that marks as zero the values that need to be inferred, i.e. the trajectories $\xi^{ij}_{[1,H]}$, and to one the other values, i.e. the imposed initial conditions.

*Model training.* The training proceeds as follows. The process begins with sampling $b$ latent variables, $\mathbf{z}_1, \ldots, \mathbf{z}_b$, from a latent space having the same shape as the desired output, in our problem $k = |Z| = |S^H|$. The training phase involves simulating the diffusion process for each sample in the training dataset. The diffusion process iteratively transforms the initial noise into a more realistic sample. At each diffusion step, we measure the discrepancy between the transformed sample and the real sample from the training dataset, our loss function. The parameters $\psi$ of the model are updated based on the gradient of such loss. By iteratively applying the diffusion process, computing the loss, and updating the model's parameters, the generative model learns to generate samples that resemble the real data. The training phase aims to find the optimal configuration of $\psi$ that minimizes the discrepancy between the generated samples and the target data distribution.

*Abstract model simulation.* Once the training is over, i.e. we have inferred the function $\epsilon_\psi$, we focus on the reverse process that allows us to generate samples in the trajectory space $S^H$ from an initial random noise sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ conditional on $\mathbf{y}^0$ that stays constant along the entire diffusion process (see Fig. 5).

## 4. Case studies

*SIR model: absorbing state.* The SIR epidemiological model describes a population divided in three mutually exclusive groups: susceptible (S), infected (I) and recovered (R). The system state at time $t$ is $s_t = (\mathsf{S}_t, \mathsf{I}_t, \mathsf{R}_t)$. The possible reactions, given by the interaction of individuals (representing the molecules of a CRN), are the following:

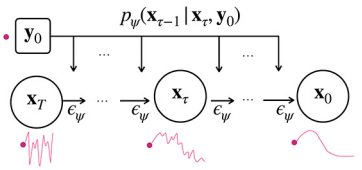


**Fig. 5.** Diagram of the CSDI architecture.

- $\mathsf{R}_1 : \mathsf{S} + \mathsf{I} \xrightarrow{\theta_1 \cdot \mathsf{I_t} \mathsf{S_t} / (\mathsf{S_t} + \mathsf{I_t} + \mathsf{R_t})} 2\mathsf{I}$ (infection),
- $\mathsf{R}_2 : \mathsf{I} \xrightarrow{\theta_2 \cdot \mathsf{I_t}} \mathsf{R}$ (recovery).

The model describes the spread, in a population, of an infectious disease that grants immunity to those who recover from it. As the SIR model is well-known and stable, we use it as a testing ground for our GAN-based abstraction procedure. The ranges for the initial state are $\mathsf{S}_0, \mathsf{I}_0, \mathsf{R}_0 \in [30, 200]$. An important aspect of the SIR model is the presence of absorbing states. In fact, when $\mathsf{I} = 0$ or when $\mathsf{R} = N$ no more reaction can take place. STL property: $\phi = F_{[t_0, t_H]} G\ (\mathsf{I} = 0)$, meaning that eventually the number of infected individuals will reach and stay at zero.

*Ergodic SIRS model: transient dynamics.* Small perturbations of the SIR model force the system to be ergodic. We called this revised version ergodic SIRS (eSIRS). This model has no absorbing state. In particular, we assume that the population is not perfectly isolated, meaning there is always a chance of getting infected by some external individuals. In addition, we also assume that immunity is only temporary. The possible reactions are now the following:

- $\mathsf{R}_1 : \mathsf{S} + \mathsf{I} \xrightarrow{\theta_1 \cdot \mathsf{I_t} \mathsf{S_t} / (\mathsf{S_t} + \mathsf{I_t} + \mathsf{R_t}) + \theta_2 \cdot \mathsf{S_t}} 2\mathsf{I}$ (infection),
- $\mathsf{R}_2 : \mathsf{I} \xrightarrow{\theta_3 \cdot \mathsf{I_t}} \mathsf{R}$ (recovery),
- $\mathsf{R}_3 : \mathsf{R} \xrightarrow{\theta_4 \cdot \mathsf{R_t}} \mathsf{S}$ (immunity loss).

Both epidemiological models are essentially unimodal. The ranges for the initial state are $\mathsf{S}_0, \mathsf{I}_0, \mathsf{R}_0 \in [0, N]$ such that $\mathsf{S}_0 + \mathsf{I}_0 + \mathsf{R}_0 = N$. In our experiments $N = 100$. The range for parameter $\theta_1$ is $[0.5, 5]$. STL property: $\phi = F_{[t_0, t_H]} G\ (\mathsf{I} \leq 25)$, meaning that from a certain time the number of infected will always stay under a safety threshold (set to 25 in our experiments).

*Genetic toggle switch model: bistability and partial observability.* The toggle switch is a well-known bistable biological circuit. Briefly, this system consists of two genes, $\mathsf{G}_1$ and $\mathsf{G}_2$, that mutually repress each other. The system displays two stable equilibrium states in which either of the two gene products represses the expression of the other gene. The possible reactions are:

- $\mathsf{prod_i} : \mathsf{G_i^{on}} \xrightarrow{\mathsf{kp_i} \cdot \mathsf{G_i^{on}}} \mathsf{G_i^{on}} + \mathsf{P_i}$, for $i = 1, 2;$,
- $\mathsf{bind_i} : 2\mathsf{P_j} + \mathsf{G_i^{on}} \xrightarrow{\mathsf{kb_i} \cdot \mathsf{G_i^{on}} \cdot \mathsf{P_j} \cdot (\mathsf{P_j} - 1)} \mathsf{G_i^{off}}$, for $i = 1, 2$ and $j = 2, 1$ resp.;
- $\mathsf{unbind_i} : \mathsf{G_i^{off}} \xrightarrow{\mathsf{ku_i} \cdot \mathsf{G_i^{off}}} \mathsf{G_i^{on}} + 2\mathsf{P_j}$, for $i = 1, 2$ and $j = 2, 1$ resp.;
- $\mathsf{deg_i} : \mathsf{P_i} \xrightarrow{\mathsf{kd_i} \cdot \mathsf{Pi}} \emptyset$, for $i = 1, 2$.

The ranges for the initial state are $\mathsf{G}^{\mathsf{on}}_{1,0}, \mathsf{G}^{\mathsf{on}}_{2,0} \in \{0, 1\}$ and $\mathsf{P}_{1,0}, \mathsf{P}_{2,0} \in [5, 20]$. STL property: $\phi = F_{[t_0, t_H]}\ G\ (\mathsf{P}_1 \leq 1) \vee F_{[t_0, t_H]}\ G\ (\mathsf{P}_2 \leq 1)$. In our experiments, we will abstract only the evolution of what are likely to be observable species, i.e. proteins $\mathsf{P}_1$ and $\mathsf{P}_2$. We thus marginalize over the information provided by the gene status, which is assumed to be unobservable.

*Oscillator model: periodicity.* The oscillator circuit consists of three species A, B and C and three reactions, in which A converts B to itself, B converts C to itself, and C converts A to itself. The three species regulate each other in a cyclic manner. This circuit was found to exhibit oscillations in the concentrations of the three species.

- $\mathsf{R}_1 : \mathsf{A} + \mathsf{B} \xrightarrow{\theta \cdot \frac{\mathsf{A} \cdot \mathsf{B}}{\mathsf{A} + \mathsf{B} + \mathsf{C}}} 2\mathsf{A}$ (B transformation),
- $R_2 : B + C \xrightarrow{\theta \cdot \frac{B \cdot C}{A + B + C}} 2B$ (C transformation),
- $\mathsf{R}_3 : \mathsf{C} + \mathsf{A} \xrightarrow{\theta \cdot \frac{\mathsf{C} \cdot \mathsf{A}}{\mathsf{A} + \mathsf{B} + \mathsf{C}}} 2\mathsf{C}$ (A transformation).

The ranges for the initial state are $\mathsf{A}_0, \mathsf{B}_0, \mathsf{C}_0 \in [20, 100]$. We define a consistency STL property $\phi = G_{[t_0, t_H]}(\mathsf{A} + \mathsf{B} + \mathsf{C} = N)$, checking that the point-specific population size $N$ is preserved along trajectories.

*Lotka Volterra: stiffness.* The predator-prey model consists of two species A, B and C and three reactions, in which prey A reproduce, predators B dies or predators eat preys. For proper reaction rates ($k_1 = k_3 = 10$ and $k_2 = 0.01$), this shows rapidly changing stiff dynamics.

- $\mathsf{R}_1 : \mathsf{A} \xrightarrow{\mathsf{k}_1 \cdot \mathsf{A}} 2\mathsf{A}$,
- $R_2 : A + B \xrightarrow{k_2 \cdot \frac{AB}{A+B}} 2B$,
- $\mathsf{R}_3 : \mathsf{B} \xrightarrow{\mathsf{k}_3 \cdot \mathsf{B}} \emptyset$.

The ranges for the initial state are $\mathsf{A}_0, \mathsf{B}_0 \in [500, 1000]$.

*MAPK model: ultra-sensitivity.* Mitogen-activated protein kinase cascade is a particular type of signal transduction into protein phosphorylation (PP) whose function is the amplification of a signal. The sensitivity increases with the number of cascade levels, such that a small change in a stimulus results in a large change in the response. Negative feedback from MAPK-PP to the MAKKK activating reaction with ultra-sensitivity to an input stimulus, governed by parameter $V_1$.

- $\mathsf{R}_1 : \mathsf{MKKK} \xrightarrow{\mathsf{V}_1 \cdot \mathsf{MKKK} / ((1 + (\mathsf{MAPK\_PP}/\mathsf{K}_\mathsf{I})^\mathsf{n}) \cdot (\mathsf{K}_1 + \mathsf{MKKK}))} \mathsf{MKKK\_P}$,
- $\mathsf{R}_2 : \mathsf{MKKK\_P} \xrightarrow{\mathsf{V}_2 \cdot \mathsf{MKKK\_P} / (\mathsf{K}_2 + \mathsf{MKKK\_P})} \mathsf{MKKK}$,
- $\mathsf{R}_3 : \mathsf{MKK} \xrightarrow{\mathsf{k}_3 \cdot \mathsf{MKKK\_P} \cdot \mathsf{MKK} / (\mathsf{K}_3 + \mathsf{MKK})} \mathsf{MKK\_P}$,
- $\mathsf{R}_4 : \mathsf{MKK\_P} \xrightarrow{\mathsf{k}_4 \cdot \mathsf{MKKK\_P} \cdot \mathsf{MKK\_P} / (\mathsf{K}_4 + \mathsf{MKK\_P})} \mathsf{MKK\_PP}$,
- $\mathsf{R}_5 : \mathsf{MKK\_PP} \xrightarrow{\mathsf{V}_5 \cdot \mathsf{MKK\_PP} / (\mathsf{K}_5 + \mathsf{MKK\_PP})} \mathsf{MKK\_P}$,
- $\mathsf{R}_6 : \mathsf{MKK\_P} \xrightarrow{\mathsf{V}_6 \cdot \mathsf{MKK\_P} / (\mathsf{K}_6 + \mathsf{MKK\_P})} \mathsf{MKK}$,
- $\mathsf{R}_7 : \mathsf{MAPK} \xrightarrow{\mathsf{k}_7 \cdot \mathsf{MKK\_PP} \cdot \mathsf{MAPK} / (\mathsf{K}_7 + \mathsf{MAPK})} \mathsf{MAPK\_P}$,
- $\mathsf{R}_8 : \mathsf{MAPK\_P} \xrightarrow{\mathsf{k}_8 \cdot \mathsf{MKK-PP} \cdot \mathsf{MAPK\_P} / (\mathsf{K}_8 + \mathsf{MAPK\_P})} \mathsf{MAPK\_PP}$,
- $\mathsf{R}_9 : \mathsf{MAPK\_PP} \xrightarrow{\mathsf{V}_9 \cdot \mathsf{MAPK\_PP} / (\mathsf{K}_9 + \mathsf{MAPK\_PP})} \mathsf{MAPK\_P}$,
- $\mathsf{R}_{10} : \mathsf{MAPK\_P} \xrightarrow{\mathsf{V}_{10} \cdot \mathsf{MAPK\_P} / (\mathsf{K}_{10} + \mathsf{MAPK\_P})} \mathsf{MAPK}$.

The ranges for the initial state are: $\mathsf{MKKK}_0, \mathsf{MKKK}_0\_P \in [0, 100]$ with $\mathsf{MKKK}_0 + \mathsf{MKKK}_0\_P = 100$; $\mathsf{MKK}_0, \mathsf{MKK}_0\_\mathsf{P}$, $\mathsf{MKK}_0\_\mathsf{PP} \in [0, 300]$ with $\mathsf{MKK}_0 + \mathsf{MKK}_0\_\mathsf{P} + \mathsf{MKK}_0\_PP = 300$; $\mathsf{MAPK}_0$, $\mathsf{MAPK}_0\_\mathsf{P}$, $\mathsf{MAPK}_0\_\mathsf{PP} \in [0, 300]$ with $\mathsf{MAPK}_0 + \mathsf{MAPK}_0\_\mathsf{P} + \mathsf{MAPK}_0\_\mathsf{PP} = 300$. The STL property is $\phi = F_{[t_0, t_H]}\ G\ (\mathsf{MAPK\_PP} \geq 100)$.

*E. coli model: multi-scale.* Model of the spatial motion of an E. coli bacterium when presented with a spatially varying nutrient field. An E. coli cell moves by operating multiple flagellum/motor (F/M) pairs, which can either drive it straight (run) or rotate it in place. The motility state of the cell is determined by the number of flagella found in particular conformations curly (C), semicoiled (S) and normal (N). The population of flagella is modelled as a stochastic bistable system. The possible transitions between flagellum/ motor states are summarised as

- $\mathsf{R}_1 : \mathsf{N} \xrightarrow{\mathsf{k}_-} \mathsf{S}$,
- $\mathsf{R}_2 : \mathsf{S} \xrightarrow{\mathsf{k}_+} \mathsf{N}$,
- $\mathsf{R}_3 : \mathsf{C} \xrightarrow{\mathsf{k}_+} \mathsf{N}$,
- $\mathsf{R}_4 : \mathsf{S} \xrightarrow{5\mathsf{s}^{-1}} \mathsf{C}$.

The transition rates $\mathsf{k}_\pm$ depend on the sensory response of the cell to external nutrients which is governed by a variable m representing the methylation state of the nutrient receptors. The stochastic evolution of m depends in turn on the concentration of nutrients L, therefore $\mathsf{k}_\pm(\mathsf{m}, \mathsf{L})$. For the mathematical details see [29].

The *multi-scale* simulation proceeds as follows. We fix a time-step for events at the higher scale $\overline{\Delta t}$ and a time-step for events at the lower scale $\Delta t$ so that $\overline{\Delta t} = H \cdot \Delta t$. The nutrient field and the response represent the higher level. The internal CRN of the F/M state is simulated for a horizon $\overline{\Delta t}$ and is thus the lower level of simulation. Trajectories of length $\overline{\Delta t}$ are checked against a property $\phi_{run} = (\mathsf{N} \geq 2) \vee (\mathsf{S} = 0)$ to decide if the cell goes straight (run) or rotates. The overall simulation time is set to 500 s. We define a consistency STL property $\phi = G_{[t_0, t_H]}(\mathsf{N} + \mathsf{C} + \mathsf{S} = 6)$, checking whether the selected number or flagella is preserved.

## 5. Experiments

In this section, we detail the experiments performed to validate our generative model abstractions on the case studies presented in the previous section.

*Experimental settings.* The workflow can be divided in steps: (1) define a CRN model, (2) generate the synthetic datasets via SSA simulation, (3) learn the abstract model by training either the cWGAN-GP or the CSDI model and, finally, (4) evaluate such abstraction over a test set, (5) fine-tune the models w.r.t a property $\phi$. All the steps have been implemented in Python building on several tools and libraries. In particular, CRN models are defined and simulated using the GillesPy2[2] library. PyTorch [30] is used to craft the desired architecture for both deep generative frameworks. All the experiments were performed on a 32-Core Processor, 64 GB of RAM and an NVidia A100 GPU with 20 GB, and 8 VCPU The source code, the synthetic datasets and the pre-trained models for all the experiments can be found at the following link: https://github.com/francescacairoli/GenerativeModels_Abstraction.git.

*Datasets.* For each case study, the training set consists of 20K different SSA trajectories. In particular, $N_{train} = 2\text{K}$ and $M_{train} = 10$. The test set, instead, consists of 25 new initial settings and from each of these we simulate 1K trajectories, so to obtain an accurate empirical approximation of the distribution targeted by model abstraction. For the MAPK case study, whose trajectories present variegate behaviours, we build a larger training set consisting of 100K SSA trajectories ($N_{train} = 2\text{K}$ and $M_{train} = 50$). We manually choose $H$ and $\Delta t$ so that the system is close to steady state at time $H \cdot \Delta t$, without spending there too many steps. The time interval should be small enough to capture the full transient behaviour of the system. For systems with no steady state, such as the oscillating models, we choose $H$ and $\Delta t$ so to observe a full period of oscillation. For the multi-scale E. coli scenario, where we abstract the low-level dynamics, we choose $H \cdot \Delta t$ to match the timestep of the dynamics at the higher scale. In practice, the chosen values are the following: SIR: $\Delta t = 0.5$, $H = 16$; e-SIRS: $\Delta t = 0.1$, $H = 32$; Toggle Switch: $\Delta t = 0.1$, $H = 32$; Oscillator: $\Delta t = 1$, $H = 32$; MAPK: $\Delta t = 60$, $H = 32$; E. coli: $\Delta t = 0.0015$, $H = 32$.

**Remark 1.** SSA realizations evolve over continuous time with exponentially distributed events. However, time must be discretized in order to feed the realizations into the generative models (in a data-driven application $\Delta t$ may be dictated by the available data). Time discretization is a critical task as it could lead to significant loss of information. To mitigate this problem, an adaptive selection of the step size could be employed or, alternatively, the fast-reacting species could be averaged out during simulation [31], replacing them with their steady-state. Nevertheless, this issue itself constitutes a research topic per se and is thus out of the scope of this paper. As a matter of fact, the reconstruction accuracy of our generative abstraction is up to the precision of the discrete data received as input. In other words, it only learns what it is allowed to observe regardless of its fidelity to the original process. If $\Delta t$ is too large to capture fast reactions the generative abstraction will not be able to reconstruct them. An interesting future work, inspired by [31], would be to separately learn the fast and slow dynamics.

*Data preprocessing.* Data have been scaled to the interval $[-1, 1]$ to enhance the performance of the two generative models and avoid sensitivity to different scales in species counts. During the evaluation phase, the trajectories have been scaled back. Hence, results and errors are shown in the original scale.

### *5.1. cWGAN-GP architectural details*

The same architecture and the same set of hyper-parameters work well for all the analyzed case studies, showing great stability and usability of the proposed solution. The Wasserstein formulation of GANs, with gradient penalty, strongly contributes to such stability. Traditional GANs have been tested as well, but they do not have such strength. The details of the architecture follows the best practice suggestions provided in [19]. Architectural details are summarized in Fig. 6. The Adam algorithm [32] is used to optimize the loss function of both the critic and the generator. The learning rate is set to 0.0001. Training times depend on the dimension of the dataset, on the size of mini-batches, on the number of species, and on the architecture of the cWGAN-GP. The latter has been kept constant for all the case studies. Batches of 256 samples have been used and the number of epochs varies from 200 to 400 depending on the complexity of the model. Moreover, each training iteration of the generator corresponds to 5 iterations of the critic, to balance the power of the two players. Leveraging the GPU acceleration, the average time required for training 50 epochs ranges from 1 to 5 minutes.

### *5.2. CSDI architectural details*

As before, the same architecture and the same set of hyper-parameters work well for all the analyzed case studies, showing great stability and usability of the proposed solution. Recall that $\epsilon_\psi : \left(\mathbb{R}^{n_{obs} \times H} \times \mathbb{R} \mid \mathbb{R}^{n_{cond}}\right) \to \mathbb{R}^{|X| \times H}$, where $n_{obs}$ and $n_{cond}$ denote the number of features of the target and of the conditioning space respectively. The diffusion horizon is set to $T = 50$, the minimum noise level $\beta_1 = 0.0001$, and the maximum noise level $\beta_T = 0.5$. In order to let $\alpha_\tau$ decay gently, which has been shown to improve the sample quality, we adopted the following quadratic schedule for other noise levels:

[2] https://github.com/StochSS/GillesPy2.git.

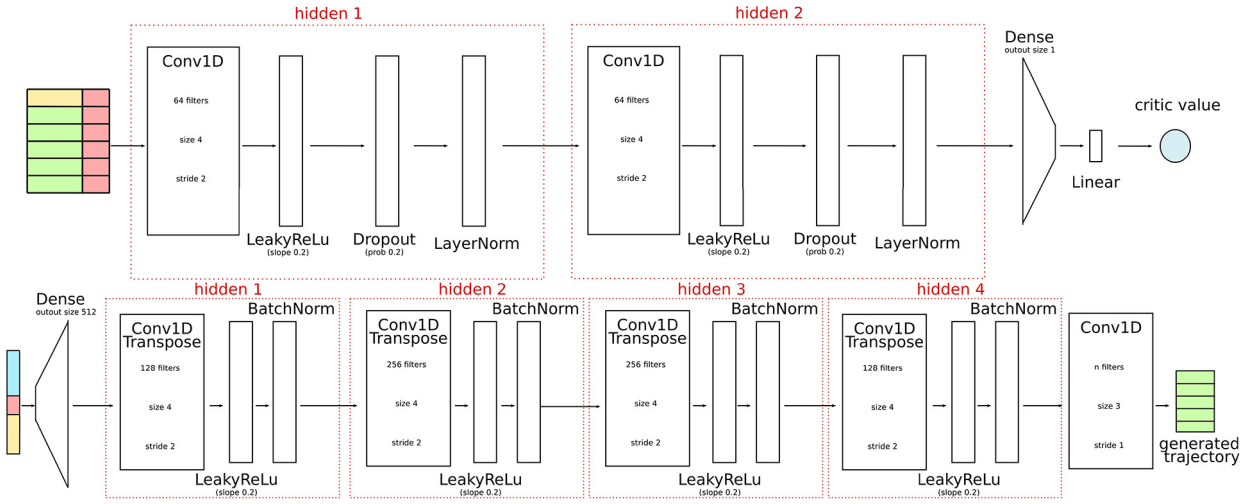


**Fig. 6.** Architectural details of the critic (top) and generator (bottom) network.

**Table 1**
Computational times required to simulate the *multi-scale* E. Coli system either via SSA, the cWGAN-GP abstraction or the CSDI abstraction.

| | SSA | cWGAN-GP | CSDI |
|---|---|---|---|
| multi-scale | 8.5 hours | 25 minutes | 55 minutes |

$$\beta_\tau = \left( \frac{T-\tau}{T-1}\sqrt{\beta_1} + \frac{\tau-1}{T-1}\sqrt{\beta_T} \right)^2 .$$

The architecture of $\epsilon_\psi$ builds on that of CSDI [33], which is composed of multiple residual layers with residual channels (Fig. 6 in Appendix E of [33] summarizes the architectural details). Above all, we highlight the use of an attention mechanism in each residual layer, instead of a convolutional one, to capture temporal and feature dependencies of multivariate time series. The temporal transformer layer takes tensors for each feature as inputs to learn temporal dependency, whereas the feature transformer layer takes tensors for each time point as inputs to learn temporal dependency (see Appendix E of [33] for a detailed description of the architecture). The learning rate is set to 0.0005, we train the models for 100 epochs over mini-batches of size 256. Leveraging the GPU acceleration, the average time required for 50 epochs of training ranges from 7 to 30 minutes.

## 6. Results

In order to evaluate the performances of our abstraction procedure we consider two important measures: the accuracy of the abstract model in reconstructing the underlying stochastic process and the computational gain compared to SSA simulation time.

*Computational gain.* The computational performances are summarized in Table 1 and 2. More precisely, Table 1 compares the simulation times for the multi-scale scenario of E. coli, where the abstract models introduce a significant computational speed-up: SSA takes 8.5 hours to simulate a single trajectory ($t_{end} = 500$ s), cWGAN-GP takes 25 minutes and CSDI takes 55 minutes. Similarly, Table 2 shows the computational speed-up in generating a single trajectory over the simpler case studies that can be interpreted as the lower scale of a multi-scale scenario (as in the E. coli example). Notice how the time needed to generate abstract trajectories does not depend on the complexity of the original system. Moreover, in both the generative approaches we use the same network architecture over all the case studies, thus the time required to generate abstract trajectories is constant. The same does not hold for the SSA trajectories, whose computational costs depend on the complexity of the model and on the chosen reaction rates. Computations are performed exclusively on a single CPU processor, to perform a fair comparison. However, the evaluation of generative models can be further sped up using GPUs, especially for large batches of trajectories, but this would have introduced a bias in their favour. It is important to stress how GPU parallelization is extremely straightforward in PyTorch whereas we found no existing implementations of Gillespie's SSA or Tau-Leaping simulation that allowed for GPU parallelization. Investigating more on how to efficiently implement such a library in an interesting future direction.

The cWGAN-GP, with a latent space of size 480, takes around 2 milliseconds (ms) to simulate a single trajectory. However, when generating batches of at least 1000 trajectories the overhead reduces and the time to generate a single trajectory

**Table 2**
Comparison of the average computational time required to simulate a *single trajectory* either via SSA (both direct or approximate methods), via the cWGAN-GP abstraction or via the CSDI abstraction.

| Model | SIR | eSIRS | TS | Osc | MAPK | E. coli |
|---|---|---|---|---|---|---|
| **SSA** (direct) | 3.000 s | 3.000 s | 3.150 s | 3.100 s | 3.200 s | 3.100 s |
| - CPU (avg over 1000): | 0.003 s | 0.003 s | 0.003 s | 0.003 s | 0.004 s | 0.0031 s |
| **SSA** ($\tau$-leaping) | 5.300 s | 5.310 s | 5.340 s | 5.330 s | 5.390 s | 5.300 s |
| - CPU (avg over 1000): | 0.0055 s | 0.0056 s | 0.0068 s | 0.0059 | 0.0151 s | 0.0054 s |
| **cWGAN-GP** | 0.002 s | 0.002 s | 0.002 s | 0.002 s | 0.002 s | 0.002 s |
| - CPU (avg over 1000): $10^{-4}\times$ | 4.300 s | 4.300 s | 4.300 s | 4.300 s | 4.300 s | 4.300 s |
| - GPU (avg over 1000): $10^{-5}\times$ | 4.200 s | 4.200 s | 4.200 s | 4.200 s | 4.200 s | 4.200 s |
| **CSDI** | 0.250 s | 0.250 s | 0.250 s | 0.250 s | 0.250 s | 0.250 s |
| CPU (avg over 1000): | 0.095 s | 0.095 s | 0.095 s | 0.095 s | 0.095 s | 0.095 s |
| GPU (avg over 1000): | 0.0045 s | 0.0045 s | 0.0045 s | 0.0045 s | 0.0045 s | 0.0045 s |

stabilizes around 42 microseconds (µs) (when using GPU). On the other hand, CSDI takes around 0.25 seconds (s) to simulate a single trajectory. The time per trajectory reduces to 0.0045 s when generating a batch of 1000 with GPU. CSDI generation takes longer (compared to cWGAN-GP) as it requires to unroll the entire diffusion process so that the higher $T$ the longer the simulation will take. Finally, with SSA the time required to simulate a single trajectory varies from 3 to 3.20 seconds. This number easily increases for more complex models or for smaller reaction rates, whereas the cost of abstract simulation stays constant. When generating batches of 1000 trajectories, starting from the same initial setting, the computational time of SSA considerably decreases in the order of 3 ms as it parallelizes the computations. We observe how, in such simple scenarios, tau-leaping does not improve the computational performances of SSA as the initial overhead still dominates the simulation time. We choose these simple case studies not for their complexity but for the variety of dynamical behaviour shown by their trajectories. The computational gain over simple case studies may seem limited, however, as we iterate the simulation process, we soon reach a significant computational speed-up, as observed in the multi-scale scenario of E. coli (Table 1).

The training phase introduces a fixed overhead that affects the overall computational gain. For instance, in cWGAN-GP training the MAPK model takes around 20 minutes (200 epochs with mini-batches of size 256) and 4 minutes (200 epochs with mini-batches of size 256) for all the other case studies, which is equivalent to the time needed to generate respectively 400 and 80 individual SSA trajectories. On the other hand, in CSDI training the MAPK model takes around 57 minutes (200 epochs with mini-batches of size 256) and 13 minutes (100 epochs with mini-batches of size 256) for all the other case studies, which is equivalent to the time needed to generate respectively 1.1K and 260 SSA trajectories. It follows that, together with the trajectories needed to generate the training set, the cost of the training procedure is paid off when we simulate at least 21K trajectories (102K for MAPK). In a typical biological multi-scale scenario in which we seek to simulate the evolution in time of a tissue containing hundreds of thousands or millions of cells, simulating also some of their internal pathways, the number of trajectories needed for the training phase becomes negligible and the training time is soon paid off.

*Measures of performance.* Results are presented as follows. For each model, we present a small batch of trajectories, both real and abstract for a randomly selected initial condition (Fig. 7-12). By visualizing these trajectories, we can qualitatively appreciate whether the abstract trajectories are similar to real ones and whether they capture the most important macroscopic behaviours. We also show the histograms of empirical distributions at time $t_H$ for each species to quantify the behaviour over all the 1K trajectories present in the test set. In all the case studies, CSDI outperforms cWGAN-GP in the quality of the produced trajectories. CSDI manages to faithfully retrace all the variegate qualitative behaviours, whereas cWGAN-GP is, in general, less precise.

*Measuring error propagation.* The reconstruction accuracy of the proposed abstraction procedure is performed on test sets consisting of 25 different initial settings. For each point, 1K SSA trajectories represent the empirical approximation of the true distribution over $S^H$. From each of these initial settings, we also simulate 1K abstract trajectories. Given a species $i \in \{1, \dots n\}$ and a time step $j \in \{1, \dots H\}$, we have the real one-dimensional distribution over $\xi_{i,j}$ and the generated abstract distribution over $\hat{\xi}_{i,j}$, where $\xi_{i,j}$ denotes the counts of species $i$ at time $t_j$ in a trajectory $\xi_{[1,H]}$. In order to quantify the reconstruction error, we compute the Wasserstein distance among the two one-dimensional distributions. By doing so, we are capable of seeing whether the error propagates in time and whether some species are harder to reconstruct than others. Error plots are shown in Fig. 13-15. In order to avoid sensitivity to different scales and make this distance more interpretable, we compute it over trajectories scaled in the interval $[-1, 1]$. We observe that, in all the case studies and for both abstract models, each species seems to contribute equally to the error and, in general, the error stays constant w.r.t. time, i.e., it does not propagate. This was a major concern in previous methods, based on the abstraction of transition kernels. In fact, in order to simulate a trajectory of length $H$ the abstract kernel has to be applied iteratively $H$ times. As a consequence, this results in a propagation of the error introduced in the approximation of the transition kernel. That said,

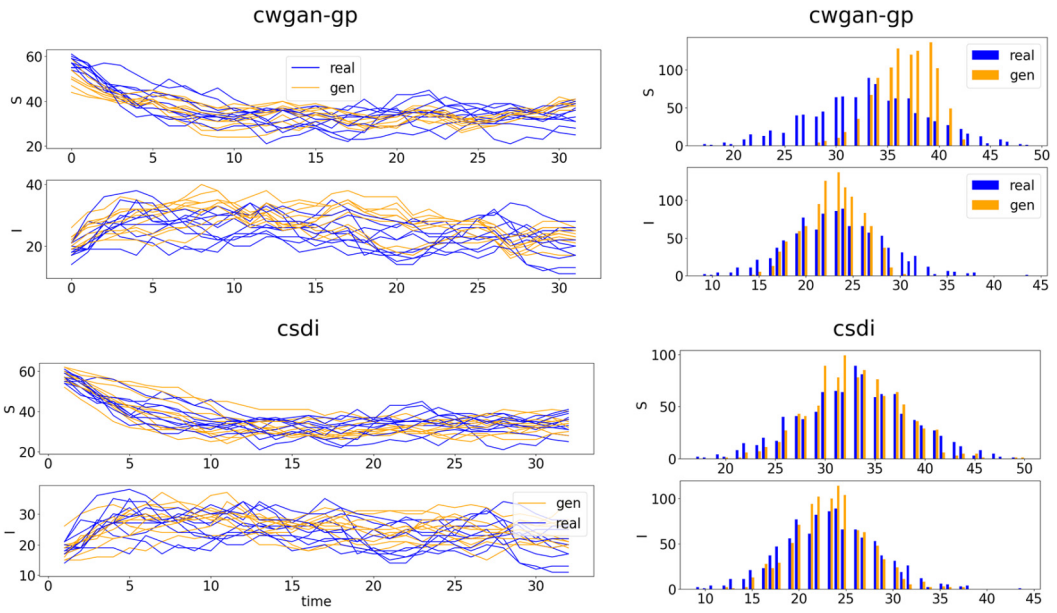


**Fig. 7. eSIRS**: on the left batch of trajectories, real (blue) and abstract (orange), for a randomly selected initial state (cWGAN-GP (top) and CSDI (top)). On the right, histograms of the trajectories distribution at time $t_H$. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

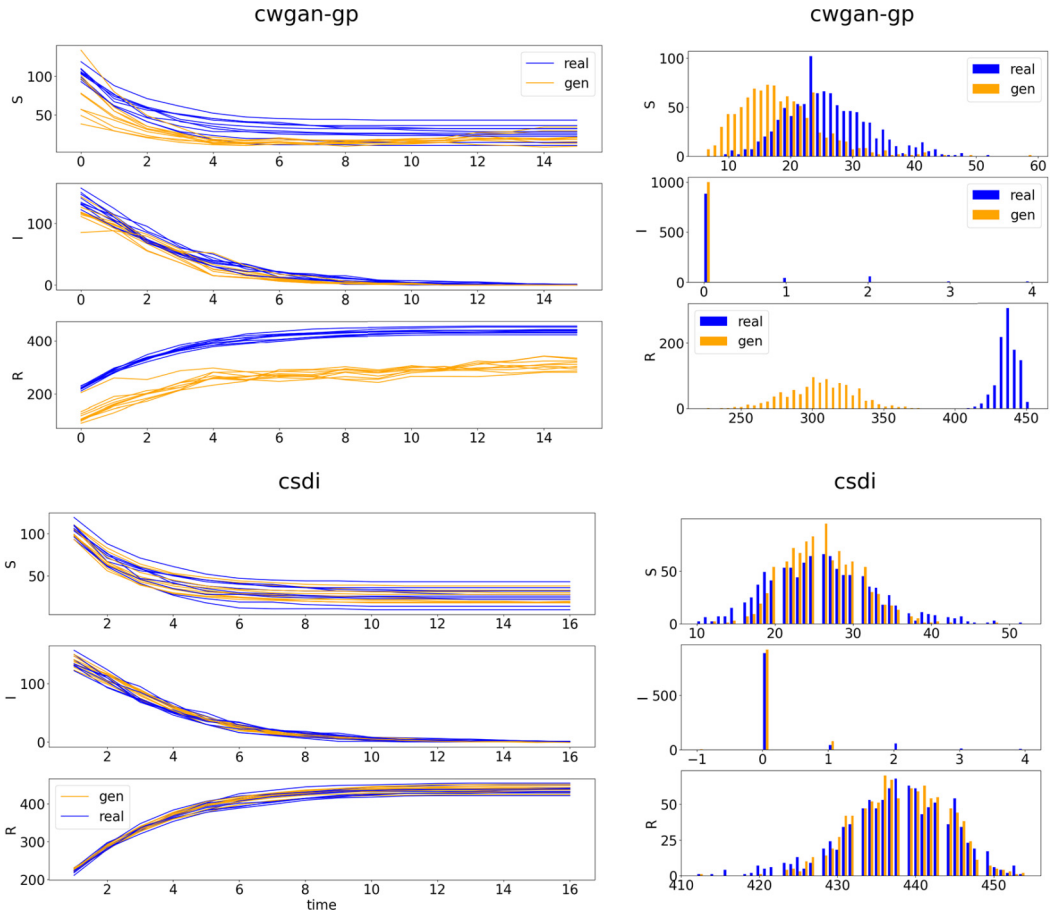


**Fig. 8. SIR**: on the left batch of trajectories, real (blue) and abstract (orange), for a randomly selected initial state (cWGAN-GP (top) and CSDI (top)). On the right, histograms of the trajectories distribution at time $t_H$.

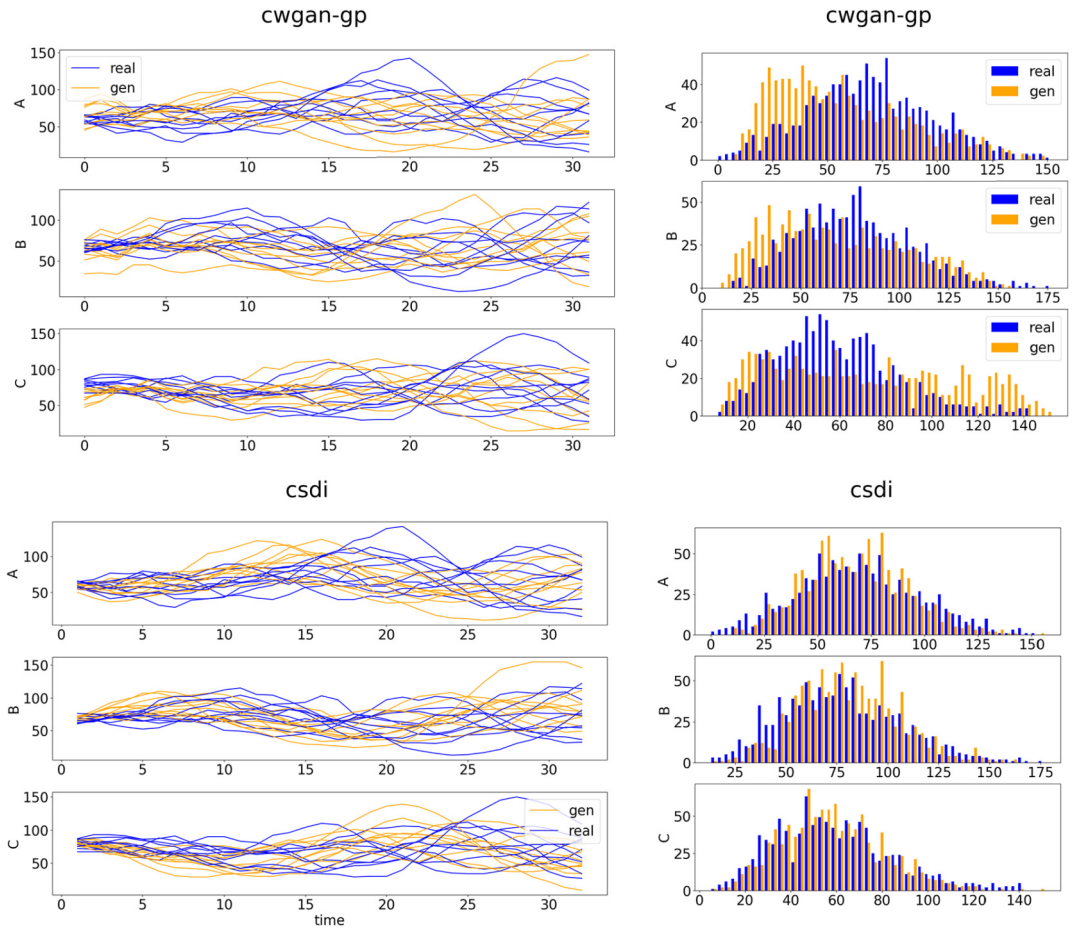


**Fig. 9. Oscillator**: on the left batch of trajectories, real (blue) and abstract (orange), for a randomly selected initial state (cWGAN-GP (top) and CSDI (top)). On the right, histograms of the trajectories distribution at time $t_H$.

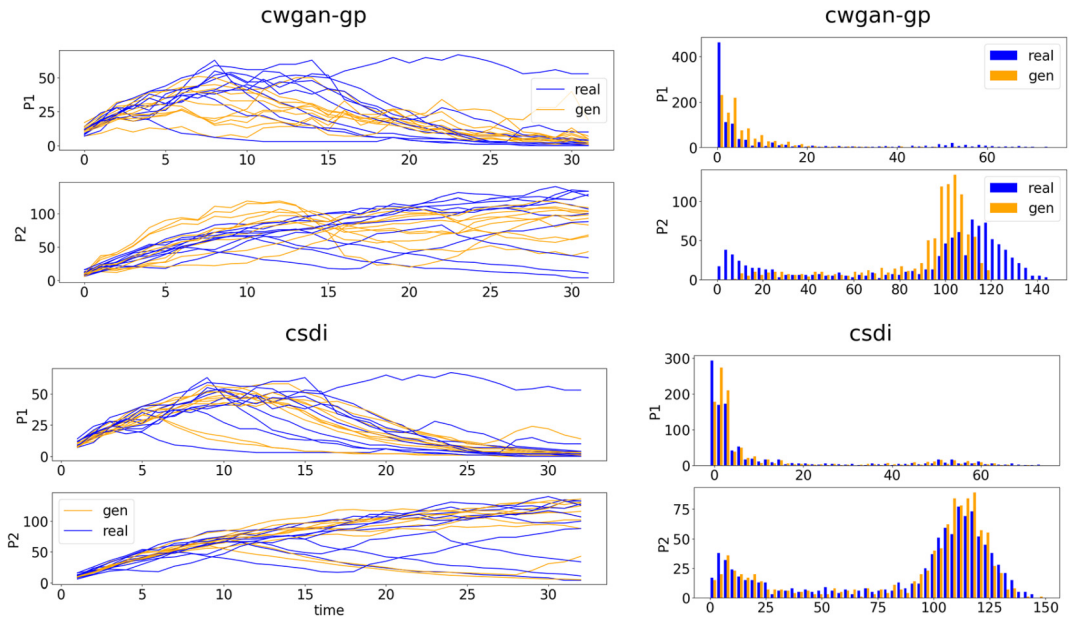


**Fig. 10. Toggle Switch**: on the left batch of trajectories, real (blue) and abstract (orange), for a randomly selected initial state (cWGAN-GP (top) and CSDI (top)). On the right, histograms of the trajectories distribution at time $t_H$.

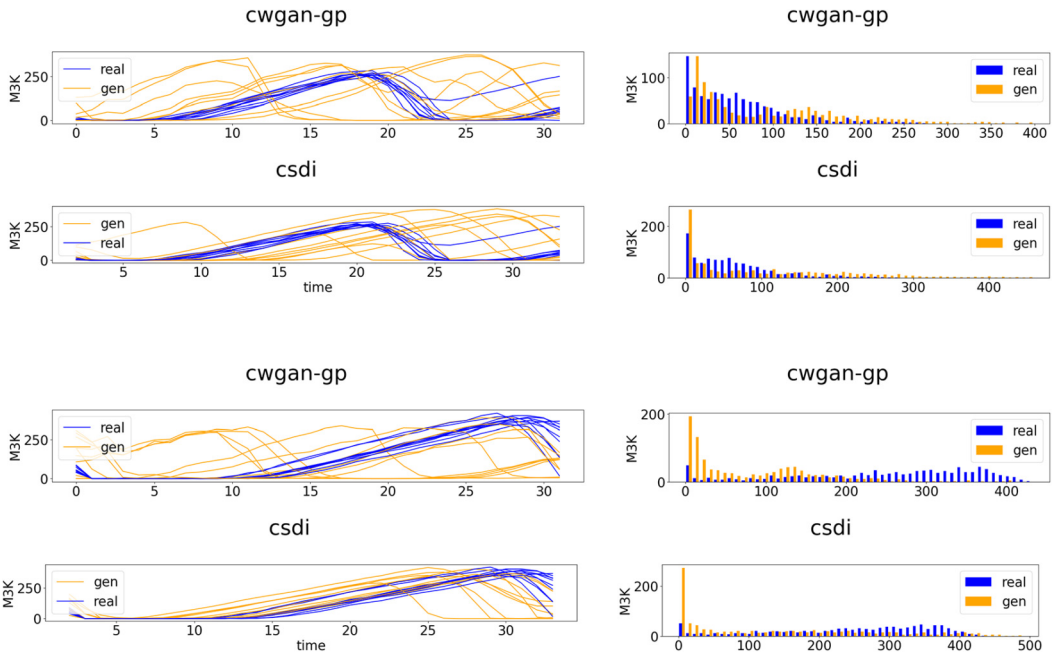


**Fig. 11. MAPK**: on the left batch of trajectories, real (blue) and abstract (orange), for two randomly selected initial states. On the right, histograms of the trajectories distribution at time $t_H$.

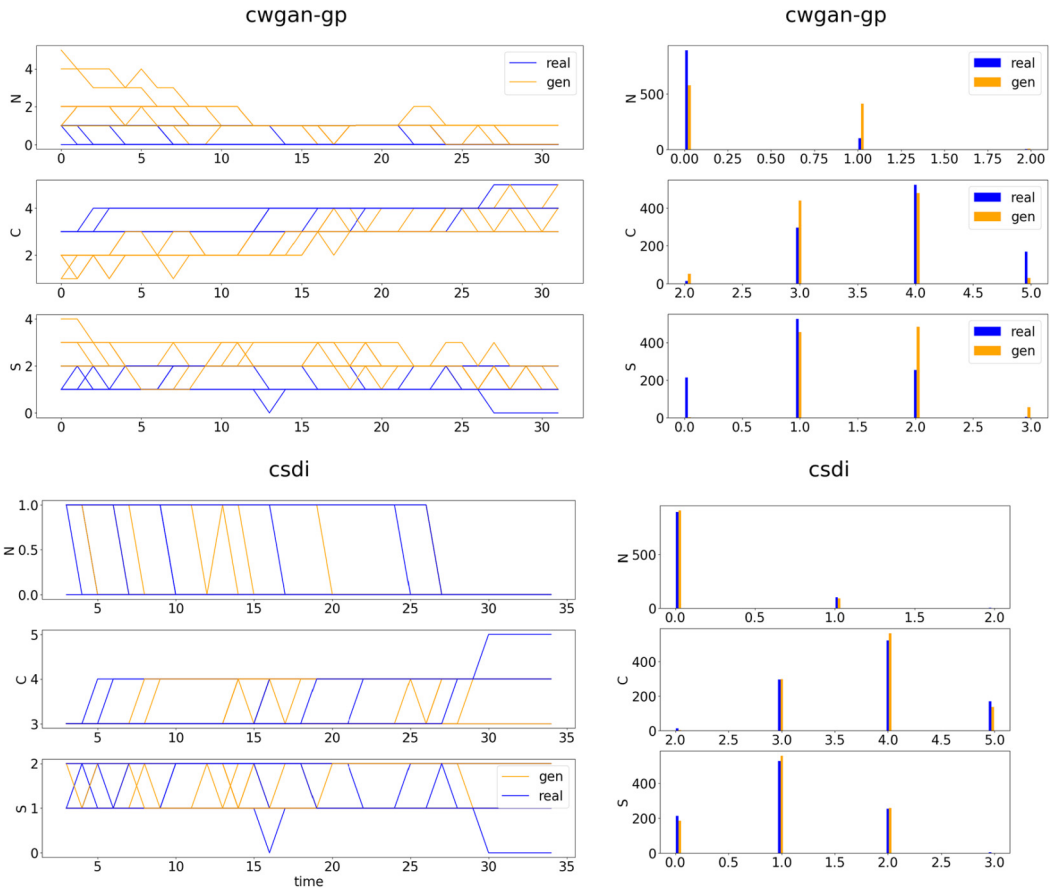


**Fig. 12. E. coli**: on the left batch of trajectories, real (blue) and abstract (orange), for a randomly selected initial state (cWGAN-GP (top) and CSDI (top)). On the right, histograms of the trajectories distribution at time $t_H$.

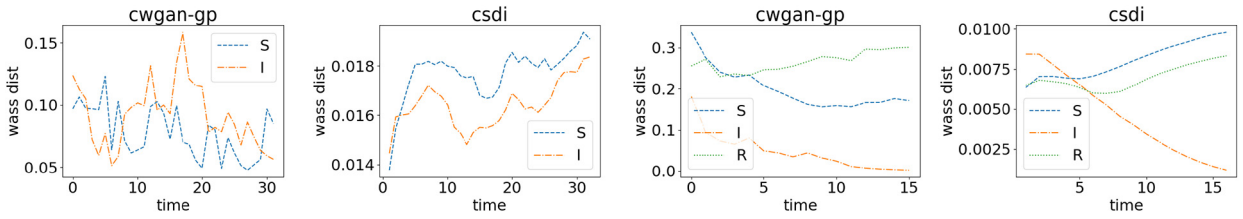


**Fig. 13. eSIRS** (left) and **SIR** (right): Wasserstein distances.

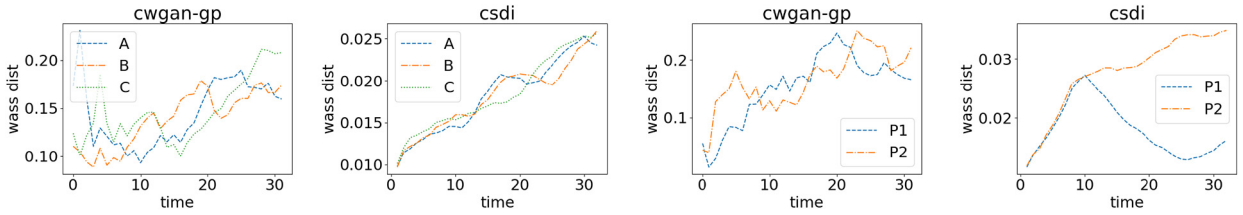


**Fig. 14. Oscillator** (left) and **Toggle Switch** (right): Wasserstein distances.

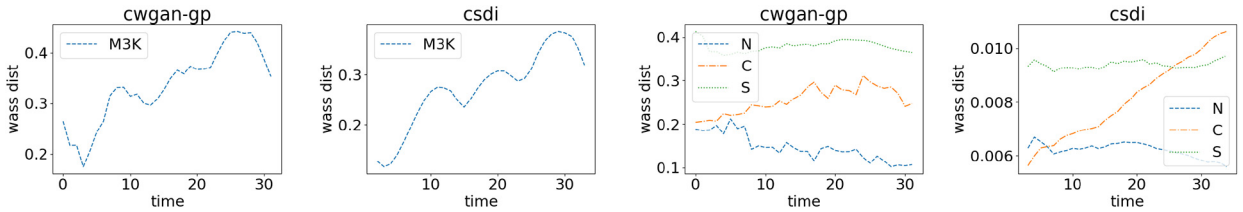


**Fig. 15. MAPK** (left) and **E. coli** (right): Wasserstein distances.

we notice how CSDI strongly outperforms cWGAN-GP in terms of accuracy as the magnitude of the error is considerably lower in all case studies.

*Choice of* $\Delta t$. In many applications, reaction rates may vary by many orders of magnitude. Rapid reactions cause species concentrations to change quickly relative to the timescales of interest introducing stiffness in the problem. Discretization over time is a critical task, especially in such scenarios, as it could lead to significant loss of information. Choosing $\Delta t$ carefully is thus essential. Our generative abstraction can reach a reconstruction accuracy up to the precision of the discrete data received as input. If $\Delta t$ is too large to capture fast reactions the generative abstraction will not be able to reconstruct them precisely. The interesting question thus becomes, whether the abstract model can capture rapidly changing stiff dynamics when they are actually shown in the data. Fig. 16 shows the estimated dynamics of a stiff Lotka-Volterra CRN for two different $\Delta t$: $\Delta t = 0.0625$ in the top row and $\Delta t = 0.25$ s in the bottom row. In both scenarios, the generative abstraction performs well, despite the inaccurate data of the second scenario.

*Level of satisfaction of STL properties.* To quantify the qualitative performance of our method, we look at the discrepancy in reconstructing the average robustness of a given STL requirement. In order to be interpretable, the STL requirements are expressed at the population level, thus trajectories have to be scaled back to the original integer values. The error values are therefore affected by the scale of the problem. We can not compare the values between different case studies but we can compare cWGAN-GP and CSDI over each specific case study, i.e. we perform a column-wise comparison in Fig. 17. Fig. 17 shows the distribution of discrepancies $\delta_i$ (Eq. (9)) for each case study and for both generative models. We notice how the score-based models strongly outperform the generative adversarial ones as all the errors show a significantly lower order of magnitude in the residual error.

*Statistical tests.* To conclude the analysis, we show the results of a two-sample statistical test over all the case studies. In particular, we use a statistical test based on the energy distance among distributions [34]. For each initial setting and for each species we compute the distance statistics and the p-value among the empirical approximation of the SSA distribution and the empirical approximation of the abstract distribution over the trajectory space, i.e. a $H$-dimensional space. Fig. 18 reports the mean and the 95% confidence interval of p-values over the initial settings present in the test set. Unsurprisingly, the p-value decreases as the number of samples used to approximate the distributions increases. Fig. 18 shows how the p-values for each species vary according to the number of samples used. Once again, CSDI outperforms cWGAN-GP as the generated empirical distribution is statistically undistinguishable from the SSA one for larger sample size (on average around 300), whereas cWGAN-GP fails in passing the test for samples of size higher than 10. These results come with no surprise

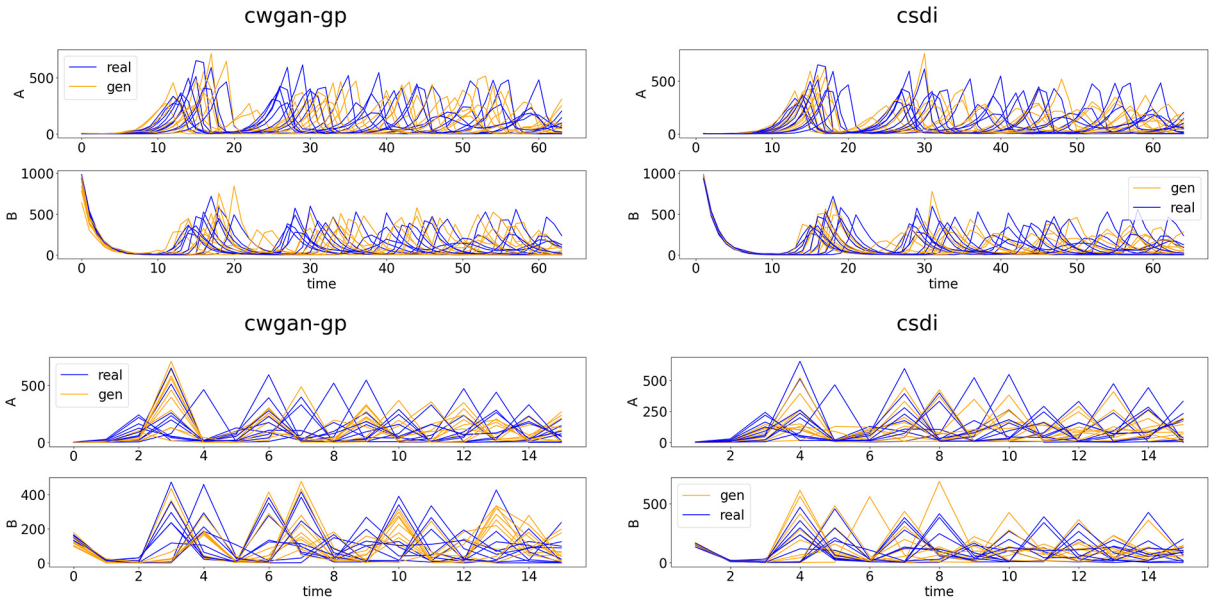


**Fig. 16.** Stiff Lotka Volterra model for different values of $\Delta t$: $\Delta t = 0.0625$ s(top) and $\Delta t = 0.25$ s (bottom).

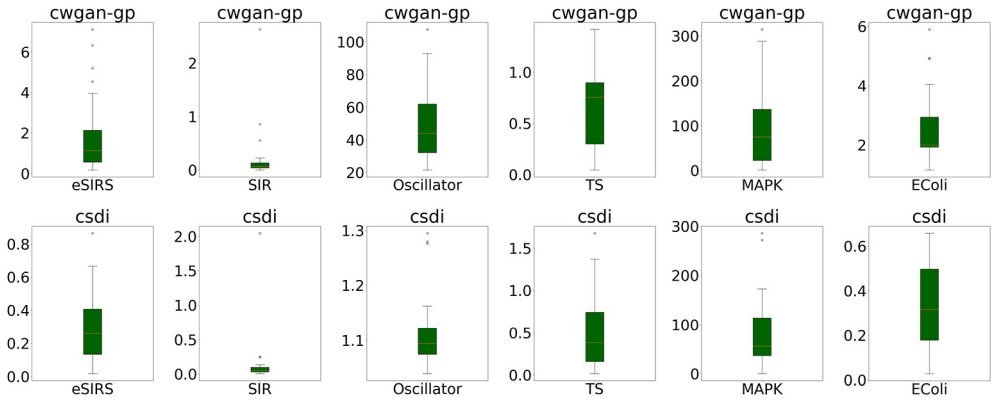


**Fig. 17.** Distribution of the error in reconstructing the average STL robustness of a requirement $\phi$ for cWGAN-GP (top row) and CSDI (bottom row) over all the case studies.

as the abstract model was trained having only 10 observations for each initial setting. In order to enhance the resilience of the abstract model to such statistical tests we should increase the number of samples per point in the training set. This comes at the cost of reducing the number of initial setting so that the resulting training set is not too large. In this regard, our active learning technique can become even more beneficial.

*Active learning.* As explained in Section 3.2, we actively select points with poor reconstruction of the average STL robustness for a property $\phi$. The query strategy is given by a GPR trained over a validation set with $N_v = 500$ initial settings and $M_v = 50$ trajectories per point. More precisely, the UCB function (extracted by the pre-trained GPR) acts as a query strategy. The actively selected points are thus used to fine-tune the abstract generators. More precisely, we run additional 100 epochs with a reduced learning rate of $10^{-7}$ (cWGAN-GP) and $10^{-8}$ (CSDI) over the selected points alone. Fig. 19 shows the results over two case studies (eSIRS and Oscillator). The error bars in dark green are exactly as those shown in Fig. 17. The bars in light green, on the other hand, show the error distribution after the finetuning. We observe how CSDI is more sensitive to the finetuning w.r.t. cWGAN-GP in both examples. As a matter of fact, the error mean of the CSDI decreases with the active fine-tuning, whereas in cWGAN-GP it depends on the case study. For instance, active learning works well for Oscillator (Fig. 17 (right)) but not for eSIRS (Fig. 17 (left)).

This active learning strategy aims to optimize the learning process by becoming more data-efficient, which is crucial when dealing with complex real-world systems. The rationale behind this approach is to start by learning an abstraction from a limited amount of measured data. By using metrics like STL satisfaction or the Wasserstein distance, it becomes possible to identify regions where the model's reconstruction is not accurate enough. To address these inaccuracies, the

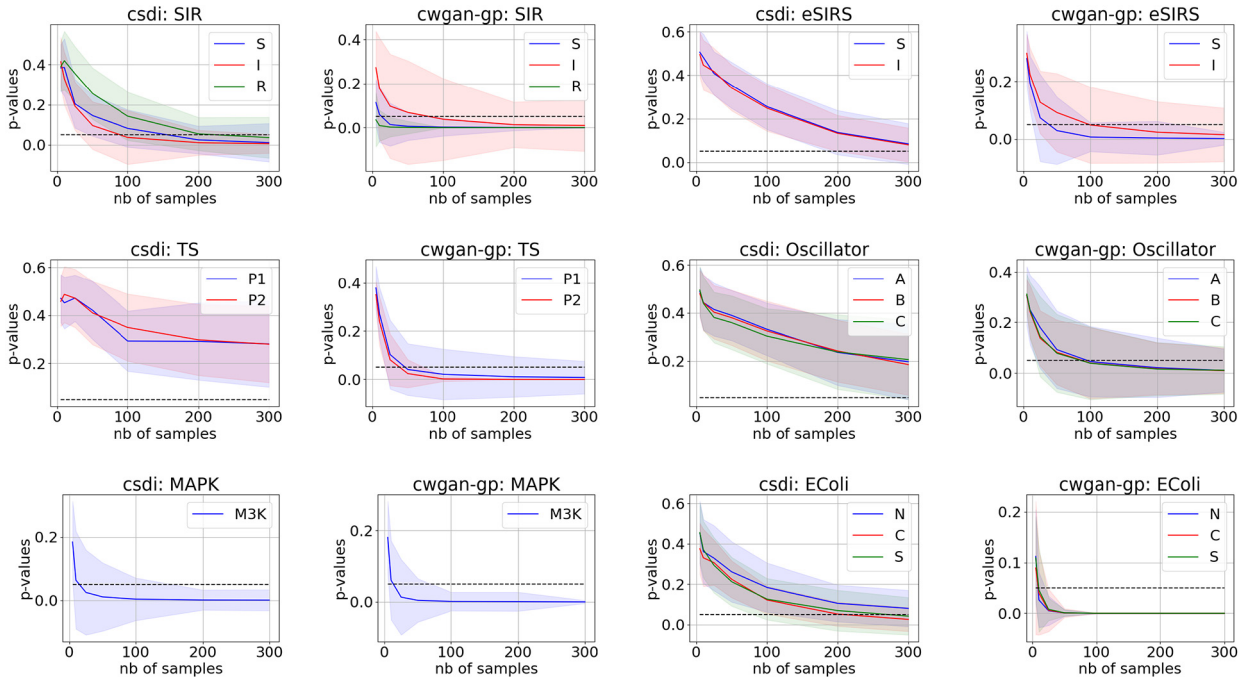


**Fig. 18.** Average over the initial setting of the p-values (with confidence interval) for each species computing by the two-sample statistical test w.r.t. the number of samples present in the empirical distributions.

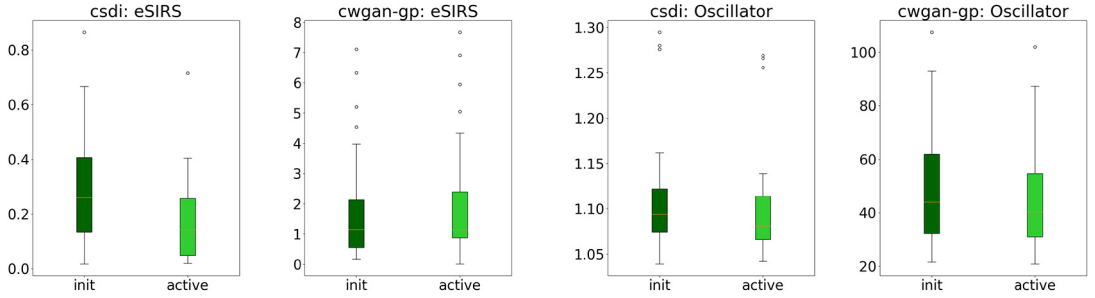


**Fig. 19.** Comparison of the distribution of error $\delta_i$ for the initial (dark green) and the actively refined (light green) generative models.

strategy involves actively collecting data specifically from these problematic regions and then retraining the abstract model. This iterative process is repeated until the reconstruction accuracy reaches a satisfactory level. As a result, active learning enables a highly effective data-collection campaign, focusing efforts where they matter the most.

**e-SIRS.** The e-SIRS model represents our baseline. The model is trained on a dataset with fixed parameters, $\theta = \{2.36, 1.67, 0.9, 0.64\}$. Results are very accurate. Fig. 7 compares the trajectories for a randomly chosen initial state represented by a pair of trajectories, the top one is for species S and the bottom one is for species I. The Wasserstein distances and STL robustnesses are presented in Fig. 13 (left) and 17 (first column).

**SIR.** Fig. 8 shows the performance on a chosen test point, cWGAN-GP is the top line and CSDI is the bottom line. Each point is represented by three trajectories, the top one is for species S, the central one is for species I and the bottom one is for species R. The population size, given by $\mathsf{S}+\mathsf{I}+\mathsf{R}$, is variable. The abstraction was trained on a dataset with fixed parameters, $\theta = \{3, 1\}$. Likewise, in the test set only the initial states are allowed to vary. We observe that both abstraction methods are able to capture the absorbing nature of SIR trajectories. It is indeed very important that once state $\mathsf{I} = 0$ or state $\mathsf{R} = N$ is reached, the system should not escape from it. Abstract trajectories satisfy such property without requiring the imposition of any additional constraint. The empirical distributions, real and generated, at time $t_H$ are almost indistinguishable. The Wasserstein distances and STL robustnesses are presented in Fig. 13 (right) and 17 (second column).

**Oscillator.** The results for the Oscillator model on a randomly chosen test points are shown in Fig. 9. The abstraction was trained on a dataset with fixed parameter ($\theta = 1$). Likewise, in the test set only the initial states are allowed to vary. Each point is represented by three trajectories, the top one is for species A, the central one is for species B and the bottom one is for species C. The abstract trajectories well capture the oscillating behaviour of the system. The Wasserstein distances and STL robustnesses are presented in Fig. 14 (left) and 17 (third column).

**Toggle Switch.** The results for the Toggle Switch model on a randomly chosen test point are shown in Fig. 10. The abstraction was trained on a dataset with fixed symmetric parameters ($kp_i = 1, kb_i = 1, ku_i = 1, kd_i = 0.01$ for $i = 1, 2$). Likewise, in the test set only the initial states are allowed to vary. In this model, we tried to abstract only trajectories of the proteins $P_1$ and $P_2$, which are typically the observable species, ignoring the state of the genes. By doing so, we reduce the dimensionality of the problem but we also lose some information about the full state of the system. Nonetheless, the generative abstractions are capable of capturing the bistable behaviour of such trajectories. Fig. 10 shows trajectories, the top one is for species $P_2$, whereas the bottom one is for species $P_2$. The Wasserstein distances and STL robustnesses are presented in Fig. 14 (right) and 17 (fourth column).

**MAPK.** The results for the MAPK model on two, randomly chosen, test points, are shown in Fig. 11. The abstraction was trained on a dataset considering only a varying $V_1$ parameter and the dynamics of species MAPK_PP. This case study represents a complex scenario in which the abstract distribution should capture the marginalization over the other seven unobserved variables. Moreover, the emergent behaviour of the only observed variable, MAPK_PP, is strongly influenced by the input parameter $V_1$ and further amplified by the multi-scale nature of the cascade: for some values of $V_1$ the system oscillates, whereas for others it stabilizes around an equilibrium. Results show that our abstraction technique is flexible enough to capture such sensitivity. The Wasserstein distances and STL robustnesses are presented in Fig. 15 (left) and 17 (fifth column).

**E. coli.** The results for the E. coli model on a randomly chosen test point are shown in Fig. 12. The abstraction was trained on a dataset considering as initial condition the initial state $s_0 = (N, C, S)$ and two varying parameters $k_+$ and $k_-$ so that it can be used to abstract the lower level dynamics of the multi-scale scenario. We are particularly interested in the quality of reconstruction at the last time step as we use this information to evolve the system at the higher scale. The histogram comparison at time $t_H$ is thus particularly insightful. The Wasserstein distances and STL robustnesses are presented in Fig. 15 (right) and 17 (sixth column).

### *6.1. Discussion*

Previous approaches to model abstraction, see related works in Section 1, focus on approximating the transition kernel, meaning the distribution of possible next states after a time $\Delta t$, rather than learning the distribution of full trajectories of length $H$. The main reason for such choice is the limited scalability of the tool used for learning the abstraction. In fact, learning a distribution over $S^H \subseteq \mathbb{N}^{H \times n}$ with a Mixture Density Network is unfeasible even for small $H$. Moreover, in learning to approximate the transition kernel one must split the SSA trajectories of the dataset into pairs of subsequent states. By doing so, a lot of information about the temporal correlation among states is lost. Other approaches, either assume the CRN structure as known or are capable of learning only one-dimensional marginal distributions, at the cost of either loosing correlation temporal among species or of learning a functional of the distribution. Having a tool strong and stable enough to learn distributions over $S^H$ allows us to preserve this information and make abstraction possible even for systems with a complex dynamics, which the abstraction of the transition kernel was failing to capture. For instance, we are now able to abstract the transient behaviour of multi-stable or oscillating systems. When attempting to abstract the transition kernel, either via MDN or via c-GAN, for such complex systems, we did not succeed in learning meaningful solutions. A collateral advantage in generating full trajectories, rather than single subsequent states, is that it introduces an additional computational speed-up in the time required to generate a large pool of trajectories of length $H$. An additional strength of our method is that one can train the abstract model only on species that are observable, reducing the complexity of the CRN model while preserving an accurate reconstruction for the species of interest. Once again, this was not possible with transition kernels and it may be extremely useful in real-world applications. Even though, the computational speed-up introduced by the generative solutions over simple models may seem limited, the proposed techniques perform well in scenarios that are very complex and challenging, e.g. multi-scale simulation. These are the scenarios that can benefit the most from such abstractions, as we may transform an unfeasible simulation problem into an approximatively solvable one. In general, the generative approximation does not provide any statistical guarantee about the reconstruction error. In addition, the set of observations used to learn the abstraction is rather small, typically 10 samples for each initial setting. However, the abstract model is actually capable of capturing, from the little amount of information provided, the emergent features of the behaviour of the original system, such as multimodality or oscillations. In this regard, we showed how formal languages can be used to formalize and check such qualitative properties. In particular, we can check whether the satisfaction probability (of non-rare events) is similar in real and abstract trajectories. Furthermore, such quantification of qualitative properties can be used to measure how good the reconstruction is. We have also shown how this qualitative comparison can be effectively used as a query strategy in an active learning approach so that the obtained abstract model is driven to the desired behaviour. To conclude, we observed how the score-based diffusion approach outperformed the Wasserstein generative adversarial one in terms of reconstruction accuracy under all the metrics at the expense of a slightly higher computational cost of simulation.

## 7. Conclusions

The paper presents two techniques to abstract the simulation process of stochastic trajectories for various Markov stochastic processes. These abstractions based on state-of-the-art deep generative models improve considerably the com-

putational efficiency, which is no more related to the complexity of the underlying process. This would be extremely helpful in all those applications in which a large number of simulations is required, i.e., applications whose solution is unfeasible via SSA simulation. It could enable the simulation of multi-scale models for very large populations, it would speed up statistical model checking [35] (as well as most of the simulation-based optimization problems) and it can be used in particular cases of parameter estimation. Moreover, the presented approach can be exploited in data-driven scenarios to learn a stochastic model from real data. This paper shows how score-based diffusion models are preferable to all the previously proposed ones, including the generative adversarial approach, as they perform considerably better under all metrics. To conclude, the generative solutions to model abstraction perform well in scenarios that are very complex and challenging, requiring very little fine-tuning. On the other hand, they may necessitate a substantial amount of data to accurately reconstruct complex dynamics, which, in turn, could limit their applicability to real-world complex systems. Nevertheless, the showcased active-learning strategy has the potential to improve data efficiency. We also stress the importance of meticulously selecting the time discretization strategy to avoid losing crucial information regarding the system's dynamics.

As future work, we plan to study how our abstraction technique works on real data. In this regard, we do not aim at capturing the underlying dynamical system, but we would rather be able to reproduce the trajectories observed in real applications. A great strength of our method, compared to previous solutions, is that it is able to generate trajectories only for a subset of the species present in the system domain, ignoring the information that is not observable, even during the training phase. Another interesting extension is to adapt our technique to impose some soft constraints over the sampled trajectories with no retraining of the generative model itself.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

We would like to thank the anonymous reviewers for their valuable suggestions and insightful comments and for the timely handling of this manuscript. This work has been partially supported by the PRIN project "SEDUCE" n. 2017TWRCNB and by the PNRR project iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043).

## References

[1] D.T. Gillespie, Exact stochastic simulation of coupled chemical reactions, J. Phys. Chem. 81 (1977) 2340–2361.
[2] Y. Cao, D.T. Gillespie, L.R. Petzold, Efficient step size selection for the tau-leaping simulation method, J. Chem. Phys. 124 (2006) 044109.
[3] J. Pahle, Biochemical simulations: stochastic, approximate stochastic and hybrid approaches, Brief. Bioinform. 10 (2009) 53–64.
[4] L. Bortolussi, L. Palmieri, Deep abstractions of chemical reaction networks, in: International Conference on Computational Methods in Systems Biology, Springer, 2018, pp. 21–38.
[5] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[6] T. Petrov, D. Repin, Automated deep abstractions for stochastic chemical reaction networks, arXiv preprint, arXiv:2002.01889, 2020.
[7] L. Bortolussi, F. Cairoli, Bayesian abstraction of Markov population models, in: International Conference on Quantitative Evaluation of Systems, Springer, 2019, pp. 259–276.
[8] A. Gupta, C. Schwab, M. Khammash, DeepCME: a deep learning framework for computing solution statistics of the chemical master equation, PLoS Comput. Biol. 17 (2021) e1009623.
[9] A. Sukys, K. Öcal, R. Grima, Approximating solutions of the chemical master equation using neural networks, iScience 25 (2022).
[10] Q. Badolle, G. Berrada, M. Khammash, Efficient Fisher information computation and policy search in sampled stochastic chemical reaction networks through deep learning, bioRxiv, 2023, 2023–04.
[11] Y. Tang, J. Weng, P. Zhang, Neural-network solutions to stochastic reaction networks, Nat. Mach. Intell. 5 (2023) 376–385.
[12] Z. Cao, R. Chen, L. Xu, X. Zhou, X. Fu, W. Zhong, R. Grima, Efficient and scalable prediction of spatio-temporal stochastic gene expression in cells and tissues using graph neural networks, bioRxiv, 2023, 2023–02.
[13] Q. Jiang, X. Fu, S. Yan, R. Li, W. Du, Z. Cao, F. Qian, R. Grima, Neural network aided approximation and parameter inference of non-Markovian models of gene expression, Nat. Commun. 12 (2021) 2618.
[14] F. Cairoli, G. Carbone, L. Bortolussi, Abstraction of Markov population dynamics via generative adversarial nets, in: Computational Methods in Systems Biology, 2021.
[15] O. Maler, D. Nickovic, Monitoring temporal properties of continuous signals, in: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer, 2004, pp. 152–166.
[16] A. Donzé, O. Maler, Robust satisfaction of temporal logic over real-valued signals, in: Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems, Springer, Klosterneuburg, Austria, 2010, pp. 92–106.
[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[18] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein GAN, arXiv preprint, arXiv:1701.07875, 2017.
[19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A.C. Courville, Improved training of Wasserstein GANs, in: Advances in Neural Information Processing Systems, 2017, pp. 5767–5777.
[20] C. Villani, Optimal Transport: Old and New, vol. 338, Springer Science & Business Media, 2008.
[21] M. Mirza, S. Osindero, Conditional generative adversarial nets, arXiv preprint, arXiv:1411.1784, 2014.
[22] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep Learning, vol. 1, MIT Press, Cambridge, 2016.

[23] J. Ho, A. Jain, P. Abbeel, Denoising diffusion probabilistic models, arXiv:2006.11239 [abs], 2020.
[24] Y. Song, S. Ermon, Generative modeling by estimating gradients of the data distribution, Adv. Neural Inf. Process. Syst. 32 (2019).
[25] Y. Song, J. Sohl-Dickstein, D.P. Kingma, A. Kumar, S. Ermon, B. Poole, Score-based generative modeling through stochastic differential equations, arXiv preprint, arXiv:2011.13456, 2020.
[26] Y. Song, S. Ermon, Improved techniques for training score-based generative models, Adv. Neural Inf. Process. Syst. 33 (2020) 12438–12448.
[27] J. Hensman, N. Fusi, N.D. Lawrence, Gaussian processes for big data, arXiv preprint, arXiv:1309.6835, 2013.
[28] J. Hensman, A. Matthews, Z. Ghahramani, Scalable variational Gaussian process classification, in: Artificial Intelligence and Statistics, PMLR, 2015, pp. 351–360.
[29] M. Michaelides, J. Hillston, G. Sanguinetti, Statistical abstraction for multi-scale spatio-temporal systems, in: Quantitative Evaluation of Systems: 14th International Conference, Proceedings 14, QEST 2017, Berlin, Germany, September 5-7, 2017, Springer, 2017, pp. 243–258.
[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: NIPS-W, 2017.
[31] L. Bortolussi, D. Milios, G. Sanguinetti, Efficient stochastic simulation of systems with multiple time scales via statistical abstraction, in: International Conference on Computational Methods in Systems Biology, Springer, 2015, pp. 40–51.
[32] Y.N. Dauphin, H. De Vries, Y. Bengio, RMSProp and equilibrated adaptive learning rates for non-convex optimization, arXiv preprint, arXiv:1502.04390v1, 2015.
[33] Y. Tashiro, J. Song, Y. Song, S. Ermon, CSDI: conditional score-based diffusion models for probabilistic time series imputation, arXiv:2107.03502 [abs], 2021.
[34] G.J. Székely, M.L. Rizzo, Energy statistics: a class of statistics based on distances, J. Stat. Plan. Inference 143 (2013) 1249–1272.
[35] H.L. Younes, R.G. Simmons, Statistical probabilistic model checking with a focus on time-bounded properties, Inf. Comput. 204 (2006) 1368–1409.