Towards Learned Simulators for Cell Migration

Anonymous Author(s)
Affiliation
Address
email

Abstract

Simulators driven by deep learning are gaining popularity as a tool for efficiently emulating accurate but expensive numerical simulators. Successful applications of such *neural* simulators can be found in the domains of physics, chemistry, and structural biology, amongst others. Likewise, a neural simulator for cellular dynamics can augment lab experiments and traditional computational methods to enhance our understanding of a cell's interaction with its physical environment. In this work, we propose an autoregressive probabilistic model that can reproduce spatiotemporal dynamics of single cell migration, traditionally simulated with the Cellular Potts model. We observe that standard single-step training methods do not only lead to inconsistent rollout stability, but also fail to accurately capture the stochastic aspects of the dynamics, and we propose training strategies to mitigate these issues. Our evaluation on two proof-of-concept experimental scenarios shows that neural methods have the potential to faithfully simulate stochastic cellular dynamics at least an order of magnitude faster than a state-of-the-art implementation of the Cellular Potts model.

1 Introduction

Studying the variety of mechanisms through which cells migrate and interact with their physical environment is of crucial importance for our understanding of cell biology. For example, cell migration plays a key role in the interaction between the immune system and implant surfaces [7, 30], the development of embryos [24], and the progression of cancer [27, 17]. As experimental capacity in the lab is inherently limited, computational methods have emerged as a tool to investigate the stochastic and dynamic movement and shape of cells. However, these methods can be computationally demanding. This is especially restrictive in scenarios requiring many simulations, for example due to substantial stochasticity or in the case of inverse design, where parameters are optimized iteratively based on the simulator's output. Moreover, parameterizing such models to realistically simulate cells can be a difficult task, requiring careful design and expert knowledge.

On the other hand, deep learning has been gaining traction as a tool for learning fast approximate simulators. For example, for continuous-time and continuous-space systems defined with partial differential equations (PDEs), neural solvers learn to emulate a system's dynamics from a dataset of simulations generated by a more computationally demanding solver [18, 28, 4, 12, 34]. In this setting, a large computational cost is paid up front to generate the training set, but once trained, approximate solutions can be generated at a fraction of the original cost. Moreover, learned simulators hold the promise of emulating systems for which the laws governing the dynamics are not known, by instead training on experimental observations.

Submitted to NeurIPS 2022 AI for Science Workshop.

Based on the above considerations, we propose to use neural simulators to simulate cellular dynamics. More specifically, we consider the scenario where both the movement and shape of the cell show stochastic aspects and are highly dynamic, which is typically modeled in the Cellular Potts modeling framework, proposed in [10]. Given their various successful applications in modeling spatiotemporal data, we hypothesize that neural simulators are capable of faithfully emulating the ground truth dynamics, while accelerating the simulation process. Our contributions are summarized as follows:

- We propose a neural simulation model to simulate stochastic single-cell dynamics similar to those generated by the Cellular Potts model;
- We develop and evaluate autoregressive training strategies, with the aim to improve the model's rollout performance and its ability to capture stochastic dynamics;
- We observe that our method has the capacity to faithfully emulate the cellular dynamics of the Cellular Potts model, while generating simulations an order of magnitude faster.

2 Background and Related Work

2.1 Cellular Potts Model

The Cellular Potts (CP) model is a computational modeling framework for simulating cellular dynamics and the dynamic and fluctuating morphology of cells on a lattice [10, 23, 1]. The CP model has gained prominence due to its flexibility in modeling cell shape and movement, the interaction between multiple cells, stochastic aspects of cell behavior, and multiscale mechanisms [25, 20, 11].

In the CP framework, the system is modeled as a Euclidean lattice L and Hamiltonian H. The function $x : L \to S$ maps each lattice site $l_i \in L$ to its state $x(l_i) \in S$, where S is the set of all cells and materials that can be present in the system. Note that in the CP literature x is commonly referred to as σ; we deviate from this to stick to machine learning convention. To evolve the system, a Markov-Chain Monte Carlo sampling algorithm is used. At every iteration, a lattice site l_i is chosen at random. Then, a proposal is made to modify x such that state $x(l_i)$ is changed to $x(l_j)$, where l_j is a site adjacent to l_i. Finally, the difference in energy ΔH is calculated between the proposed and current system state. If $\Delta H \leq 0$, the proposed state is accepted as the new system state; if $\Delta H > 0$, it is accepted with probability $e^{\frac{-\Delta H}{T}}$, with T being the *temperature* parameter of the model.

The Hamiltonian H itself differs per application, but typically consists of at least contact energy and volume preservation terms, as originally proposed in [10]:

$$H = \underbrace{\sum_{l_i, l_j \in \mathcal{N}(L)} J\left(x(l_i), x(l_j)\right)\left(1 - \delta_{x(l_i), x(l_j)}\right)}_{\text{contact energy}} + \underbrace{\sum_{c \in C} \lambda_V \left(V(c) - V^*(c)\right)^2}_{\text{volume preservation}} + H_{\text{other}}. \tag{1}$$

Where $\mathcal{N}(L)$ is the set of all pairs of neighboring lattice sites in L, $J\left(x(l_i), x(l_j)\right)$ is the contact energy between cells and/or materials $x(l_i)$ and $x(l_j)$, and $\delta_{x,y}$ is the Kronecker delta. Furthermore, C is the set of all cells in the system, $V(c)$ is the number of lattice sites occupied by cell c (from here on referred to as the *volume* of cell c), $V^*(c)$ is the target volume of cell c, and λ_V is a Lagrange multiplier. H_{other} can consist of many extensions and modifications of the original Hamiltonian, for example taking into account cellular dynamics induced by forces, gradients in chemical concentrations, cell surface area constraints, and many more biological concepts. The specific Hamiltonians used for simulating our data can be found in Appendix A.

2.2 Neural Simulators

Neural networks have been employed for simulation in many domains [19, 5], often by either combining ML models with existing numerical solvers [32, 15] or by using ML models to simulate dynamics in their entirety [4, 18, 28]. The latter, which we refer to as *neural simulators*, encompass the type of model proposed in this work, as we seek to emulate the CP simulations as a whole. Of

particular interest are autoregressive methods operating on a spatial grid, as these fit both the temporal and spatial component of the CP simulations. This setup generally comes with challenges of ensuring prediction quality and stability over longer rollout trajectories. Common approaches to address this include injecting noise and incorporating model rollouts in the training procedure [28, 4].

In the context of cellular dynamics, neural networks have been used in various settings to aid in simulation. TrajectoryNet [31] utilizes optimal transport in combination with continuous normalizing flows to interpolate time-evolving gene expression data based on population measurements at fixed timepoints. In a similar spirit, CellBox [35, 13] simulates molecular biological processes with neural ODEs [6]. LEUP [2] models cells as atomic particles moving in two-dimensional space to investigate collective cell migration. However, these methods do not address simulation of spatiotemporal dynamics of cells on a grid. Another branch of related research focuses on grid-based generative modeling of cell morphology and subcellular organization [8, 9, 22]. Very recently, Wiesner et al. [33] proposed a method to model time evolving cell shapes. Although these methods can impressively perform (conditional) cell image generation, they do not address migration dynamics.

3 Method

3.1 Machine Learning Formulation

We consider the problem of learning an autoregressive probabilistic mapping to simulate spatiotemporal cellular dynamics. More formally, at each time t the system is described by its state x^t, a categorically-valued function on a fixed grid (see also Section 2.1). Correspondingly, a system evolution is specified by a sequence of states $x^{0:t}$. We postulate a ground-truth probability distribution $p^*(x^{0:t})$ over system evolutions $x^{0:t}$ from which we can sample using the CP simulator. Hence, our aim is to learn the parameters θ of the model p_θ such that

$$\mathbb{E}_{x^{0:t} \sim p^*} \log p_\theta(x^{0:t}) \tag{2}$$

is maximized. By the Markov property of the data generating process of the CP model configurations that we consider, and with $p^*(x^0)$ a specified distribution of initial conditions, this is equivalent to maximizing Equation 3 with respect to θ:

$$\mathbb{E}_{t \sim U[0,t-1]} \mathbb{E}_{x^t \sim p^*} \log p_\theta(x^{t+1}|x^t). \tag{3}$$

Consequently, we aim to model $p^*(x^{0:t})$ by learning $p_\theta(x^{t+1}|x^t)$ and applying it autoregressively.

3.2 Model Design

The design choices for modeling $p_\theta(x^{t+1}|x^t)$ are driven by the goals to produce a model that is capable of producing realistic trajectories with good sample efficiency. To achieve high sample efficiency, we introduce latent variable z, following the structure of a Conditional Variational Autoencoder (CVAE) [26]. In our case, latent variable z is conditioned on previous state x^t according to conditional prior $p_\theta(z|x^t)$, and the subsequent state is specified by the output distribution $p_\theta(x^{t+1}|x^t, z)$.

As our model is an autoregressive latent variable model, it consists of a *forward model* that evolves the state of the system to the next state. However, rather than directly producing the parameters

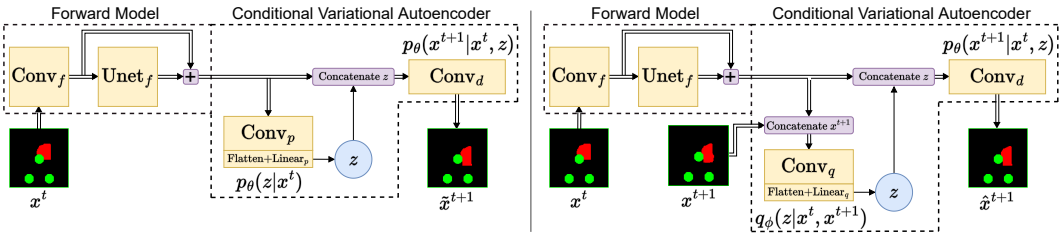

Figure 1: Illustrations of the model's generative procedure (left) and inference procedure (right). Note that the geometry of the system is maintained throughout the model, as indicated by the double line.

of a distribution that we can sample the next state from, the state representation produced by the forward model is used to first condition the prior distribution of the latent variable. Then, to produce a sample of the next state, we sample the latent variable from this prior and combine it with the forward representation, which is then decoded into the next state. This procedure is depicted in Figure 1 (left).

To enable the model to produce realistic trajectories, we align the model's structure with that of the data. Consequently, we maintain the geometry of the system in the forward representations, which in practice means using a fully convolutional architecture in the forward model as well as the decoder. Specifically, the forward model is implemented using the U-net architecture [21], but it could in principle be substituted for another fully convolutional architecture. Additionally, we do not push all information through a latent variable that has no geometrical structure, but allow the decoder to also access the intermediate, geometrically meaningful representation produced by the forward model.

The use of a U-net is further motivated by the desire to model dynamics on various spatial scales. Other architectural details are provided in Appendix B. Note that, as mentioned before, latent variable z does not share the system geometry. To sidestep this issue z is broadcasted to the entire domain before being concatenated to the internal representation, as done in a probabilistic U-net [16].

3.3 Training and Rollout Stability

As per the VAE framework, an approximate posterior distribution $q_\phi(z|x^t, x^{t+1})$ is learned, also referred to as the inference network. Note that this distribution is inferred from both the current and next timestep. Then, while training, we optimize the Evidence Lower Bound (ELBO) on the log-likelihood as follows:

$$\log p_\theta(x^{t+1}|x^t) \geq -KL(q_\phi(z|x^t, x^{t+1})||p_\theta(z|x^t)) + \mathbb{E}_{q_\phi(z|x^t,x^{t+1})}[\log p_\theta(x^{t+1}|x^t, z)]. \quad (4)$$

The inference procedure is depicted in Figure 1 (right). During trajectory generation the subsequent timestep is not available and we infer latent variable z only from the prior distribution $p_\theta(z|x^t)$. By then sampling $\tilde{x}^{t+1} \sim p_\theta(x^{t+1}|x^t, z)$ and repeating the entire process one can simulate over longer timeframes.

One of the major challenges of autoregressive neural simulators is the instability of long rollouts. As the model is applied iteratively, errors accumulate, causing $p_\theta(x^{t+r}|x^t)$ to stray from the data distribution for sufficiently large rollout length $r \in \mathbb{N}^+$. To mitigate this issue, various methods have been proposed, typically involving some form of noise injection or rollout training. We consider two training methods.

The **Multi-step** training procedure applies the model for r iterations and calculates the loss for each iteration. At each step t, a reconstruction $\hat{x}^{t+1}$ is created by sampling from $p_\theta(\hat{x}^{t+1}|x^t, z)q_\phi(z|x^t, x^{t+1})$. $\hat{x}^{t+1}$ then serves as input for the encoder and processor, which is used to create a reconstruction $\hat{x}^{t+2}$, and so on. Note that, since $\hat{x}^{t+1}$ is discrete, the gradients are only backpropagated a single step. The **Pushforward** training method, adapted from [4], is similar in nature. We sample $r \sim U[1, r_{\max}]$, rollout the model r times as described for multi-step training, but now only calculate and backpropagate the loss at the final iteration. In this way, the model learns to correct its own rollout errors and to map back to the data distribution. From a probabilistic perspective, the rollout errors can be seen as adding noise to the data, where the noise is sampled from the model's error distribution itself.

Orthogonal to the training procedure, we also consider two ways to sample from p_θ to improve stability. Both have in common that z is sampled from the posterior (when training) or the prior (when generating new simulations), but differ in how $\tilde{x}^{t+1}$ is sampled from $p_\theta(x^{t+1}|z, x^t)$. With **maximum likelihood** sampling, a single z is sampled, and $\tilde{x}^{t+1}$ is discretized such that $p_\theta(x^{t+1}|z, x^t)$ is maximized for that z. However, this method does not exploit any domain knowledge about the system. With **volume preservation** sampling, knowledge about the (approximately) preserved volume of the cell is injected in the method. Here, $\tilde{x}^{t+1}$ is also discretized to maximize $p_\theta(x^{t+1}|z, x^t)$, but under the additional constraint that the volume of the cell equals its target volume.

4 Experiments

Our evaluation consists of two experiments. In the first, randomly scattered walls are placed on a grid, and a force pointing to the bottom center of the lattice is applied to the cell. Despite the comparatively simple setting, local stochasticity in the system already leads to interesting behavior such as cell shape fluctuations, and even emerging bifurcations of trajectories.

The second experiment consists of a system where more global stochasticity with bifurcating trajectories is simulated. A downwards force is applied to the cell, such that it makes contact with pillars placed at fixed positions along its path. Then, either a leftward or rightward force is applied to the cell with equal probability, such that it passes each pillar on the left or right. The system always has approximately identical initial conditions, but a strongly multi-modal distribution over trajectories.

All Cellular Potts simulations are generated with CompuCell3D [29], the state-of-the-art software package for CP model simulations. We implement python extensions in order to generate the desired dynamics. In all experiments, one time unit corresponds to 500 Monte Carlo Steps in CompuCell3D. For each experiment, we use 1350 training, 150 validation and 150 testing trajectories.

We compare maximum likelihood and volume preservation sampling and investigate various training strategies, see Section 3.3 for details on both. All models are trained for 300 epochs. We optimize the ELBO with the reparameterization trick [14] and use a linear KL-annealing schedule [3]. We set the maximum rollout length $r_{\max}$ to a third of the total trajectory length for pushforward and multi-step training.

4.1 Simple Dynamics

Quantitatively, we evaluate models with two metrics: the estimated ELBO on the log-likelihood (LL), and the *unrolled reconstruction log-likelihood* (URLL). LL is calculated as the sum of the lower bounds on the one-step conditional log-likelihoods (see also Equations 3 and 4). URLL is a heuristic metric for assessing rollout stability. Starting from initial state x^0, we estimate $p_\theta(x^1|x^0)$, and at each subsequent step, the log-likelihood of x^t is estimated from $p_\theta(x^t|\hat{x}^{t-1})$, where $\hat{x}^{t-1} \sim p_\theta(\hat{x}^{t-1}|\hat{x}^{t-2}, z)q_\phi(z|\hat{x}^{t-2}, x^{t-1})$ is a sampled *reconstruction* of the model at the preceding timestep. A high URLL should correspond to a model that can reconstruct entire trajectories well, rather than only individual steps.

The results of simulators with varying training and sampling strategies are given in Table 1. As expected, training strategies such as pushforward and multi-step training improve the URLL score for both sampling methods, compared to single-step training. Interestingly, pushforward training is not always beneficial for single-step log-likelihoods, whereas multi-step training does show an improvement. Another interesting result is that volume preservation sampling does not necessarily improve upon maximum likelihood sampling. We expected that the former would net an advantage as domain knowledge is integrated in the sampling procedure, but the results are not conclusive. In fact, multi-step training with maximum likelihood sampling performs best for both metrics, albeit with a small margin.

Qualitatively, the simulators can capture the stochastic behavior of the system, both locally in the form of a fluctuating cell membrane, and globally, generating diverse but realistic trajectories from identical

Training strategy	Sampling method	LL	URLL
One-step	Maximum likelihood	$\geq$ -3211.9	$\geq$ -40669.8
	Volume preservation	$\geq$ -4049.3	$\geq$ -100243.9
Pushforward	Maximum likelihood	$\geq$ -6973.3	$\geq$ -36920.8
	Volume preservation	$\geq$ -3874.3	$\geq$ -12975.8
Multi-step	Maximum likelihood	$\geq$ **-2751.0**	$\geq$ **-4619.9**
	Volume preservation	$\geq$ -3031.5	$\geq$ -4689.1

Table 1: Evaluation of training and sampling strategy combinations (best values marked in bold).

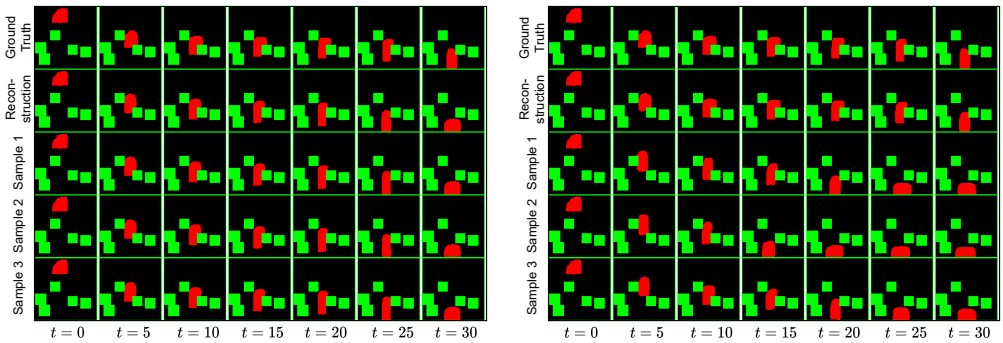

Figure 2: Ground truth and sampled trajectories, using volume preservation sampling and one-step (left) and multi-step (right) training. The cell is depicted in red, walls are depicted in green.

initial conditions. However, this behavior is not captured well when using single-step training. This is exemplified in Figure 2, which shows reconstructed and sampled trajectories for models using volume preservation, trained with one-step and multi-step training. Despite the probabilistic approach, one-step training results in samples that are almost identical. In contrast, the same architecture trained with multi-step training is capable of generating a variety of realistic trajectories. As a sanity check, we also tried replacing the CVAE by a deterministic model, which mapped the forward model's output directly to $\tilde{x}^{t+1}$. As expected, such an approach could not capture any stochastic dynamics, resulting in a frozen cell and averaged out trajectories.

We also measured the time required for one model rollout of 60 steps (corresponding to 30000 MCS for the CP model simulation) on commodity hardware (details in Appendix C). Over 100 repetitions, the average model rollout time was 2.16 seconds ($\sigma = 0.08s$) on our CPU, and 0.56 seconds ($\sigma = 0.01s$) on our GPU. Generating a simulation with CompuCell3D took 14.16 seconds on average ($\sigma = 0.76$) on the same CPU, and GPU acceleration is not supported by CompuCell3D for simulations not involving chemical diffusion. Consequently, even for these simple dynamics, the neural simulator already provides an 85% speedup compared to the CP model on identical hardware. Moreover, we envision speedups to become even larger when the dynamics are more involved, for example, dynamics involving diffusion of chemicals or multi-cellular systems.

4.2 Bifurcating dynamics

Figure 3 shows trajectories for bifurcating dynamics with identical initial conditions. Observe that the model generates trajectories in which the cell follows varying realistic paths. To quantitatively assess the training strategies, we examine the position of the cell at the end of the simulation. If the neural simulator works well, these aggregate statistics should match those of the trajectories generated by

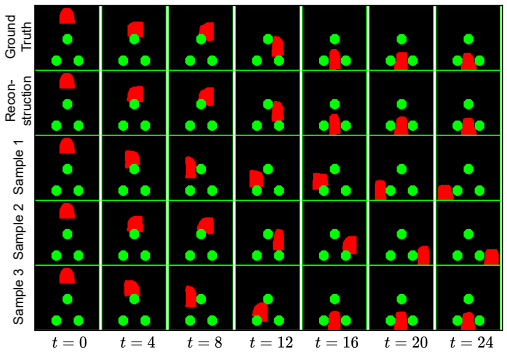

Figure 3: Trajectories generated by the CP model and the volume-preserving model with multi-step training.

Training method	Left	Center	Right
One-step	0.27	0.72	0.01
Pushforward	0.03	0.54	0.43
Multi-step	0.25	0.52	0.23
Cellular Potts	0.23	0.5	0.27
Expected	0.25	0.5	0.25

Table 2: Empirical distribution of the x-coordinate of the cell's center of mass at the end of the simulation. Trajectories are sampled from the volume-preserving model trained with various training strategies, and from the Cellular Potts model.

the CP model. Here, we present volume-preserving models as we found them to work better in this scenario; results for models using maximum likelihood sampling are provided in Appendix D. The end positions' distributions are found in Table 2. The expected distribution closely matches that of the CP model and our method trained with multi-step training. However, when trained with one-step or pushforward training, the distribution of the neural simulator differs substantially from the true distribution. This demonstrates that a probabilistic model in itself is not sufficient to capture global, long-term stochastic aspects of the evolution of the system, and training methods that go beyond single-step predictions are necessary.

5 Conclusion

In this work, we proposed a probabilistic neural simulation model for spatiotemporal cellular dynamics. We adapted training strategies from autoregressive models and found that these improve rollout quality and enable the model to accurately capture the stochastic dynamics of the system. To evaluate our method, we generated data using the CP model and show that the learned simulator is capable of faithfully emulating the dynamics. Furthermore, sampling from the learned simulator is around an order of magnitude faster than the CP simulations that generated the training data. We conclude that neural simulators are a promising method for simulating spatiotemporal cellular dynamics, with many interesting avenues for relevant research, for example simulating systems with multiple cells, or integrating neural cellular dynamics simulators and neural PDE solvers for multiscale modeling.

References

[1] Ariel Balter, Roeland M. H. Merks, Nikodem J. Popławski, Maciej Swat, and James A. Glazier. *The Glazier-Graner-Hogeweg Model: Extensions, Future Directions, and Opportunities for Further Study*, pages 151–167. Birkhäuser Basel, Basel, 2007.

[2] Arnab Barua, Josue M. Nava-Sedeño, Michael Meyer-Hermann, and Haralampos Hatzikirou. A least microenvironmental uncertainty principle (leup) as a generative model of collective cell migration mechanisms. *Scientific Reports*, 10(1):22371, Dec 2020.

[3] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21. ACL, 2016.

[4] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[5] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, 2020.

[6] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[7] B. Ann Dalton, X. Frank Walboomers, Mark Dziegielewski, Margaret D. M. Evans, Sarah Taylor, John A. Jansen, and John G. Steele. Modulation of epithelial tissue and cell migration by microgrooves. *Journal of Biomedical Materials Research*, 56(2):195–207, 2001.

[8] Rory M. Donovan-Maiye, Jackson M. Brown, Caleb K. Chan, Liya Ding, Calysta Yan, Nathalie Gaudreault, Julie A. Theriot, Mary M. Maleckar, Theo A. Knijnenburg, and Gregory R. Johnson. A deep generative model of 3d single-cell organization. *PLOS Computational Biology*, 18(1):1–24, 01 2022.

[9] Peter Goldsborough, Nick Pawlowski, Juan C Caicedo, Shantanu Singh, and Anne E Carpenter. Cytogan: Generative modeling of cell images. *bioRxiv*, 2017.

[10] François Graner and James A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Phys. Rev. Lett.*, 69:2013–2016, Sep 1992.

[11] Tsuyoshi Hirashima, Elisabeth G. Rens, and Roeland M. H. Merks. Cellular potts modeling of complex multicellular behaviors in tissue morphogenesis. *Development, Growth & Differentiation*, 59(5):329–339, 2017.

[12] Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time PDEs from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2021.

[13] Weiqi Ji, Bo Yuan, Ciyue Shen, Aviv Regev, Chris Sander, and Sili Deng. Inference of cell dynamics on perturbation data using adjoint sensitivity. In *ICLR 2021 SimDL workshop*, 2021.

[14] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[15] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[16] Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R. Ledsam, Klaus Maier-Hein, S. M. Ali Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[17] Sandeep Kumar, Aastha Kapoor, Sejal Desai, Mandar M. Inamdar, and Shamik Sen. Proteolytic and non-proteolytic regulation of collective cell invasion: tuning by ecm density and organization. *Scientific Reports*, 6(1):19905, Feb 2016.

[18] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[19] Frank Noé, Alexandre Tkatchenko, Klaus-Robert Müller, and Cecilia Clementi. Machine learning for molecular simulation. *Annual Review of Physical Chemistry*, 71(1):361–390, 2020. PMID: 32092281.

[20] Elisabeth G. Rens and Leah Edelstein-Keshet. From energy to cellular forces in the cellular potts model: An algorithmic approach. *PLOS Computational Biology*, 15(12):1–23, 12 2019.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[22] Xiongtao Ruan and Robert F Murphy. Evaluation of methods for generative modeling of cell and nuclear shape. *Bioinformatics*, 35(14):2475–2485, 12 2018.

[23] Nicholas J. Savill and Paulien Hogeweg. Modelling morphogenesis: From single cells to crawling slugs. *Journal of Theoretical Biology*, 184(3):229–235, 1997.

[24] Elena Scarpa and Roberto Mayor. Collective cell migration in development. *Journal of Cell Biology*, 212(2):143–155, January 2016.

[25] M. Scianna and L. Preziosi. Multiscale developments of the cellular potts model. *Multiscale Modeling & Simulation*, 10(2):342–382, 2012.

[26] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

[27] Catalina-Paula Spatarelu, Hao Zhang, Dung Trung Nguyen, Xinyue Han, Ruchuan Liu, Qiaohang Guo, Jacob Notbohm, Jing Fan, Liyu Liu, and Zi Chen. Biomechanics of collective cell migration in cancer progression: Experimental and computational methods. *ACS Biomaterials Science & Engineering*, 5(8):3766–3787, 2019.

[28] Kim Stachenfeld, Drummond Buschman Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation. In *International Conference on Learning Representations*, 2022.

[29] Maciej H. Swat, Gilberto L. Thomas, Julio M. Belmonte, Abbas Shirinifard, Dimitrij Hmeljak, and James A. Glazier. Chapter 13 - multi-scale modeling of tissues using compucell3d. In *Computational Methods in Cell Biology*, volume 110 of *Methods in Cell Biology*, pages 325–366. Academic Press, 2012.

[30] Thomas Thenard, Anita Catapano, Michel Mesnard, and Rachele Allena. A cellular potts energy-based approach to analyse the influence of the surface topography on single cell motility. *Journal of Theoretical Biology*, 509:110487, September 2020.

[31] Alexander Tong, Jessie Huang, Guy Wolf, David van Dijk, and Smita Krishnaswamy. Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[32] Kiwon Um, Robert Brand, Yun (Raymond) Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[33] David Wiesner, Julian Suk, Sven Dummer, David Svoboda, and Jelmer M Wolterink. Implicit neural representations for generative modeling of living cell shapes. *arXiv preprint arXiv:2207.06283*, 2022.

[34] Tailin Wu, Takashi Maruyama, and Jure Leskovec. Learning to accelerate partial differential equations via latent global evolution. *arXiv preprint arXiv:2206.07681*, 2022.

[35] Bo Yuan, Ciyue Shen, Augustin Luna, Anil Korkut, Debora S. Marks, John Ingraham, and Chris Sander. Cellbox: Interpretable machine learning for perturbation biology with application to the design of cancer combination therapy. *Cell Systems*, 12(2):128–140.e4, 2021.

A Hamiltonians for Simple and Bifurcating Dynamics

The Hamiltonian for both simple and bifurcating dynamics (Sections 4.1 and 4.2) is as follows:

$$
\begin{aligned}
H = & \underbrace{\sum_{l_i, l_j \in \mathcal{N}(L)} J\left(x(l_i), x(l_j)\right)\left(1 - \delta_{x(l_i), x(l_j)}\right)}_{\text{contact energy}} \\
& + \underbrace{\lambda_v \left(V(c) - V^*(c)\right)^2}_{\text{volume constraint}} \\
& + \underbrace{\lambda_a \left(A(c) - A^*(c)\right)^2}_{\text{surface area constraint}} \\
& + \underbrace{{\lambda_{\mathbf{F}}}^T \cdot \mathbf{COM}(\mathbf{c})}_{\text{external potential}}
\end{aligned}
$$

Where $\mathbf{COM}(\mathbf{c})$ is the center-of-mass vector of the cell c.

The values for each of the parameters for both simple and bifurcating dynamics are given in table 3:

	T	$J(\text{cell}, \text{medium})$	$J(\text{cell}, \text{wall})$	λ_V	$V^*(c)$	λ_A	$A^*(c)$	$\lambda_{\mathbf{F}}$
Simple	15	8	10	5	500	1	70	$[F_x^{\text{simple}}, -35]^T$
Bifurcating	15	8	16	5	500	1	70	$[F_x^{\text{bifurcate}}, -35]^T$

Table 3: CP parameters for both experiments

Here, $F_x^{\text{simple}} = \frac{50 - COM(c)_x}{50} \cdot 35$ is continuously updated throughout the CP simulation. For the bifurcating dynamics, if the cell is not in contact with a micropillar or if the cell reached the bottom of the grid, $F_x^{\text{bifurcate}} = 0$. If the cell comes into contact with a micropillar and $F_x^{\text{bifurcate}}$ equals 0, it is set to a value with magnitude equal to the contact area with a micropillar, of which the sign is negative or positive with equal probability. Finally, if the cell is still in contact with a micropillar and $F_x^{\text{bifurcate}}$ is nonzero, the horizontal force is rescaled such that its magnitude equals the contact area of the micropillar. We explicitly note that these parameters were not necessarily chosen to be biologically plausible, but to generate stylized CP simulations that exhibit behavior which will be relevant for modeling more complex, realistic scenarios as well.

B Model Architecture Details

The details of all model components are given below. All convolutional layers are 2D convolutions with kernel size 3x3, unless otherwise mentioned.

- **Forward Model:** The forward model starts with a linear convolutional layer to lift the input to a higher dimensional representation. This representation is processed by a U-net. The U-net has 4 contrastive blocks, one block that operates on the lowest level of the U-net, and 4 expansive blocks. Each contrastive block consists of two convolutional layers with ReLU activation, followed by 2x2 maxpooling. The block that operates on the lowest level has the same architecture, but does not do maxpooling. Each expansive block consists of a 2x2 upconvolution and the concatenation of the result with the output of the corresponding contrastive block. Then, two convolutional layers with ReLU activation are applied. Each contrastive block doubles the channel dimension, while each expansive block halves the channel dimension. Finally, the output of the U-net is cropped from the center such that the dimensionality of the output is the same as the input, to which it is summed (i.e., a residual connection).

- **Conditional Variational Autoencoder:** The CVAE consists of a prior network, generative network, and inference network. For the CVAE, the kernel size of the convolutional layers is 5x5 instead of 3x3.
  - Prior network $p_\theta(z|x^t)$: first, the image channel corresponding to the one-hot encoding of the walls is concatenated with the forward model's outputs. Then, two convolutional layers with ReLU activation are applied, followed by 2x2 maxpooling. Subsequently, two repetitions of a convolutional layer with ReLU activation followed by maxpooling are applied. Finally, the output is flattened and mapped to a lower-dimensional space using a linear layer with ReLU activation. Two separate linear layers map this output to the parameters of the normal distribution over the latent space μ and $\log \sigma$.
  - Generative network $p_\theta(x^{t+1}|z, x^t)$: z is concatenated channel-wise to the forward model's output, along with the wall channel of x^t. Two convolutional layers with ReLU activation are applied, followed by one convolutional layer with sigmoid activation, to obtain the pixel-wise Bernoulli probabilities of the cell's location. As the cell cannot be located on top of a wall, we explicitly set the decoder output probabilities for these pixels to 0.
  - Inference network $q_\phi(z|x^{t+1}, x^t)$: the architecture of the inference network is identical to the prior network, with the exception that it takes an extra input channel, corresponding to the cell channel of the one-hot encoding of x^{t+1}.

C Hardware

Model training was done on a single Nvidia RTX 3060 GPU, and took at most about three hours per model, depending on the dataset and training strategy. For a fair comparison of the simulation times of the Cellular Potts model and our method, we resorted to local hardware, as CompuCell3D did anyhow not support GPU acceleration for simulations without chemical diffusion. The CPU used was an Intel i7-9750H CPU clocked at 2.6GHz. For the GPU time measurements, we used a Nvidia Quadro P2000.

D Maximum Likelihood Sampling for Bifurcating Dynamics

To quantitatively evaluate maximum likelihood sampling we again investigate the cell's position at the end of a simulation. Table 4 contains the distributions of these end positions. While the model trained using the multi-step method approaches the ground-truth distribution, there are clear deviations. The volume-preserving model discussed in Section 4.2 much closer aligns with the Cellular Potts model and the expected distribution.

Training method	Left	Center	Right
One-step	0.04	0.35	0.61
Pushforward	0.41	0.57	0.02
Multi-step	0.34	0.53	0.13
Cellular Potts	0.23	0.5	0.27
Expected	0.25	0.5	0.25

Table 4: Empirical distribution of the x-coordinate of the cell's center of mass at the end of the simulation. Trajectories are generated with the model that uses maximum likelihood sampling and various training strategies, and the Cellular Potts model.