Path-minimizing Latent ODEs as Inference Models

Matt L. Sampson
Princeton University
matt.sampson@princeton.edu

Peter Melchior
Princeton University
peter.melchior@princeton.edu

Abstract

Latent ODE models provide flexible descriptions of dynamic systems, but they can struggle with extrapolation and predicting complicated non-linear dynamics. The Latent ODE approach implicitly relies on encoders to identify unknown system parameters and initial conditions, whereas the evaluation times are known and directly provided to the ODE solver. This dichotomy can be exploited by encouraging *time-independent* latent representations. By replacing the common variational penalty in latent space with an ℓ_2 penalty on the path length of each system, the models learn data representations that can easily be distinguished from those of systems with different configurations. We demonstrate superior results for simulation-based inference of the Lotka-Volterra parameters and initial conditions by using the latents as data summaries for a conditional normalizing flow.

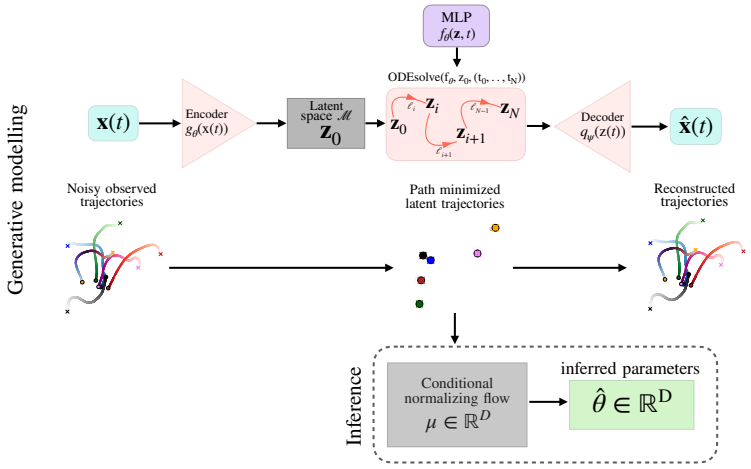

Figure 1: Schematic of our path-length minimizing latent ODE model.

1 Introduction

Latent ODEs are a class of generative neural network models for sequential data, which have seen widespread adoption as flexible empirical descriptions of dynamic systems [Chen et al., 2018, Rubanova et al., 2019]. A common architecture design uses a sequence model, such as a Recurrent Neural Network (RNN) [Rumelhart and McClelland, 1987] to initially encode an observed time series $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. The latent representation $\mathbf{z} \in \mathbb{R}^d$ is then evolved via numerical integration of the changes $\mathrm{d}\mathbf{z}/\mathrm{d}t = f_\theta(t)$, represented by a neural network with parameters θ, from the initial time t_i to the endpoint t_f. It is customary to turn the neural ODE model into a Variational Autoencoder (VAE) [Kingma and Welling, 2013] to encourage a d-dimensional Gaussian distribution in latent space, but

Machine Learning and the Physical Sciences Workshop, NeurIPS 2024.

other distributions have also been shown to work well [Rubanova et al., 2019]. However, as we not require sampling of compatible trajectories and will take measures to prevent the collapse of the latent distribution, we impose an alternative penalty: the total length of the latent trajectory. Doing so drastically reduces the temporal variation of the latent representations, so that the architectures targets the parameters of the dynamical model, not its state. We will show adding a path-length penalty to the loss improves simplicity and trainability of the models while producing more accurate predictions over a longer time frame allowing for improved inference of system parameters and initial conditions.

2 Implementation

We adopt the architecture from Rubanova et al. [2019], which uses an ODE-RNN as the recognition network, trained as VAE. We use a feed-forward neural network for the ODE function f_θ and the ODE solvers from the `diffrax` package [Kidger, 2021] for numerical integration (full details in Appendix A). However, as our primary interest lies in accurate forecasts and inference for given system configurations instead of fast sampling, we remove the variational aspect from the model, replacing the Gaussianity penalty with a path-length penalty:

$$\mathcal{L} = \underbrace{\sum_{i=1}^{n} (\mathbf{x}_i - \tilde{\mathbf{x}}_i)^2}_{\text{reconstruction loss}} + \underbrace{\lambda S}_{\text{min path loss}}, \tag{1}$$

where $\mathbf{x}_i \in \mathbb{R}^f$ denotes an observation of the system at time t_i, $\tilde{\mathbf{x}}_i$ its reconstruction by the latent ODE model, and n the number of such observations. The path-length penalty S is given by the Mahalanobis distance between successive points $\mathbf{z}_i$ and $\mathbf{z}_{i+1}$ in latent space:

$$S = \sum_{i}^{m-1} \sqrt{(\mathbf{z}_i - \mathbf{z}_{i+1})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{z}_i - \mathbf{z}_{i+1})} \tag{2}$$

The hyperparameter m represents the number of interpolation points within the timespan of the observed trajectory (note this can be much greater than n), λ controls the strength of the distance penalty. We search for best performance with a wide sweep, but we find good results for all tests with $\lambda \approx 1$ (see Appendix B for more details). To prevent the latent distribution from collapsing to a point, which trivially satisfies the path-length penalty, we scale the penalty with the empirical standard deviation in latent space $\boldsymbol{\Sigma}$ of batches of trajectories from different system configurations. This diagonal matrix gets updated after every training step.

3 Experiments

As a *baseline* we adopt the latent ODE-RNN architecture described in Rubanova et al. [2019] as well as an ODE-GRU encoder, and compare it to our implementation on the Lotka-Volterra preditor-prey model. Full details about data generation are in Appendix A, while training, hyper-parameters, and model parameters for all test cases are detailed in Appendix B. We note our models are smaller than that used in similar works [Chen et al., 2018, Rubanova et al., 2019, Shi and Morris, 2021, Coelho et al., 2024, Auzina et al., 2024]. We compare our results on recognition networks of the same size (with the exception of the HBNODE trials). For all trials, we allow the baseline model to train for up to 10 times as long as the best performing path-minimized model and report results for the best trained model.

The Lotka-Volterra equations (LVE) are a nonlinear set of coupled first-order ODEs

$$\frac{dx}{dt} = \alpha x - \beta xy, \quad \frac{dy}{dt} = \delta xy - \gamma y, \tag{3}$$

that are often used to describe the interdependence of the numbers of predators x and their prey y. The LVEs have been found to present difficulties for latent ODE type models [Shi and Morris, 2021, Auzina et al., 2024], especially their long-term behavior, and therefore present a good test case for our purposes.

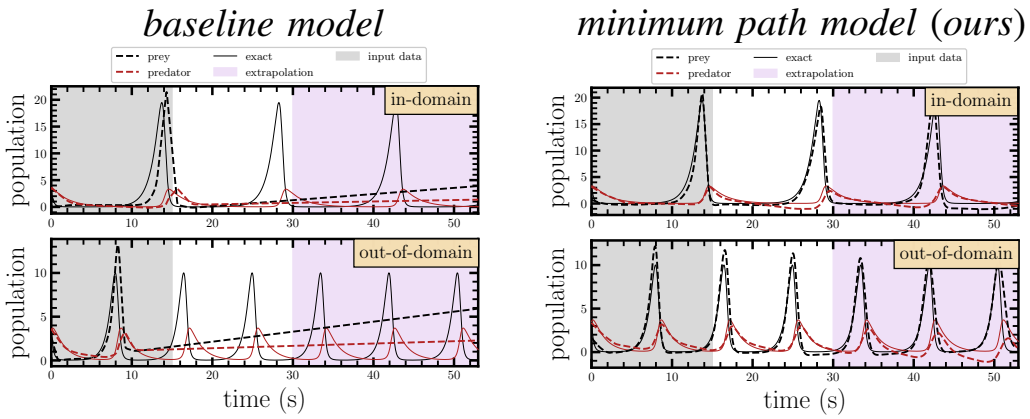

Figure 2: Reconstructions of solutions for the Lotka-Volterra equations with randomly sampled initial conditions and model parameters from within the training ranges (*top*) and from up to 25% beyond the training ranges (*bottom*). The grey shading indicates the region where data is supplied, the purple shading indicates extrapolation regions, i.e no model has seen training data past this point.

We use sample uniformly parameters $\alpha \in [1.0, 3.5]$, $\beta \in [1.0, 3.5]$, $\delta \in [0.5, 0.6]$, $\gamma \in [0.5, 0.6]$ and initial conditions $x_0, y_0 \in [1, 6]$ as in [Auzina et al., 2024]. We plot two example trajectories, one *in-domain* and one *out-of-domain* with their analytic solution in Figure 2. For the out-of-domain data, we move each of the 4 parameters 25% outside of their respective training ranges. The baseline model fares well in reconstructing in the interpolation regions ($t = 0 \to 25$), with increasing errors at longer times, for the in-domain test. Our model on the right shows high accuracy even up to large times, showing the expected cyclic behavior of the LVE. It maintains high levels of accuracy even for out-of-domain testing. Summary statistics are listed in Table 1 with trails run on both ODE-RNN and ODE-GRU enocoders, as well as a standalone comparison to the state-of-the-art HBNODE implementation for this problem Xia et al. [2021].

Table 1: Test data MSE and its standard deviation from 256 trials for the predator-prey system of our model and the baseline latent ODE-RNN. We show results for interpolation ($t = 25$) and extrapolation ($t = 50$) for in-domain and out-of-domain parameters.

t_{final}	encoder	Baseline model	Minimum path
		⟨ MSE ⟩ (std)	⟨ MSE ⟩ (std)
25	ODE-RNN	13.96 (0.57)	**5.19 (0.32)**
25	ODE-GRU	16.16 (4.12)	**12.18 (2.92)**
25	HBNODE	**2.851 (0.49)**	N/A
50	ODE-RNN	27.87 (0.86)	**12.507 (0.54)**
50	ODE-GRU	17.49 (3.77)	**113.06 (2.71)**
50	HBNODE	440.6 (24.6)	N/A

3.1 Parameter inference

The top panel of Figure 4 visualizes the distribution of latents with our path-length penalty. We show a UMAP [McInnes et al., 2018] projection of the LVE trajectories in latent space for a selection of trials (shown in different colours). As expected, the addition of a path-length penalty shortens the trajectories in our model compared to the baseline by an average factor of ≈ 30.

As a result, despite the time-dependent nature of dynamical systems, the ODE integrator only fills small regions of latent space, leaving most of its volume free to capture the time-invariant descriptors of the system: parameters and initial conditions. This behavior becomes clear when we fill the UMAP with latents from our model for 10,000 randomly selected LVE system parameters and color-code by the parameter values. The bottom panels of Figure 4 show evident structure, which means that specific parameters are located in specific locations in latent space, and that variations of parameters leads to smooth changes of latent space positions.

Table 2: Inference accuracy (relative MSE of the posterior mean and its standard deviation in 256 trials) for the LVE parameters.

pts	in-dist	Baseline	Minimum path
		⟨ rel. MSE ⟩ (std)	⟨ rel. MSE ⟩ (std)
5	Yes	1.879 (1.88)	**0.320 (0.22)**
10	Yes	0.708 (0.72)	**0.261 (0.24)**
20	Yes	0.301 (0.30)	**0.197 (0.20)**
5	No	2.045 (1.97)	**0.482 (0.31)**
10	No	0.930 (1.2)	**0.406 (0.34)**
20	No	0.464 (0.39)	**0.347 (0.25)**

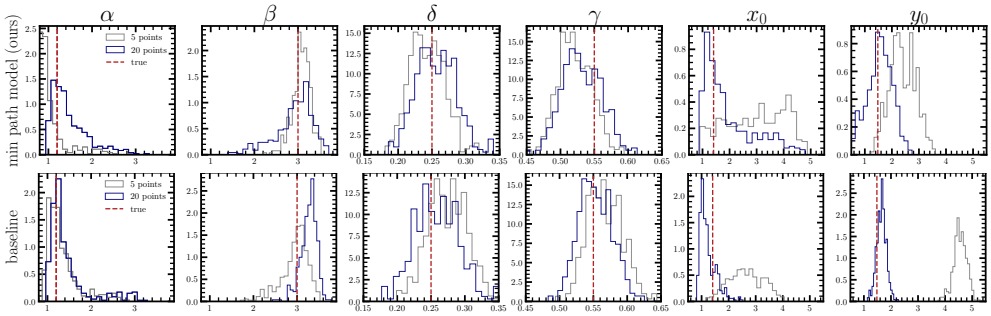

Figure 3: Posterior plots from an LVE test case using a normalizing flow to infer the value of the parameters and initial conditions, for our model (*top*) and the baseline model (*bottom*). The gray histogram shows the posteriors for $n = 5$, the blue for $n = 20$ observations. The red dashed line indicates the true parameter value.

Posteriors with simulation-based inference The evident structuring of the latent space to follow primarily parameter and IC values should allow for accurate inference from noisy and irregularly sampled observations even without the explicit splitting of static and dynamic variables proposed by Auzina et al. [2024].

To test the inference capabilities of our model, we train a normalizing flow [Rezende and Mohamed, 2015] to predict four parameters and two ICs of Equation 3 given the latent context vector from the autoencoder as an input [Winkler et al., 2019]. Details on this implementation are given in Appendix B. In the language of simulation-based inference, we use the latent ODE encoder to produce summaries for potentially irregularly sampled sequence data.

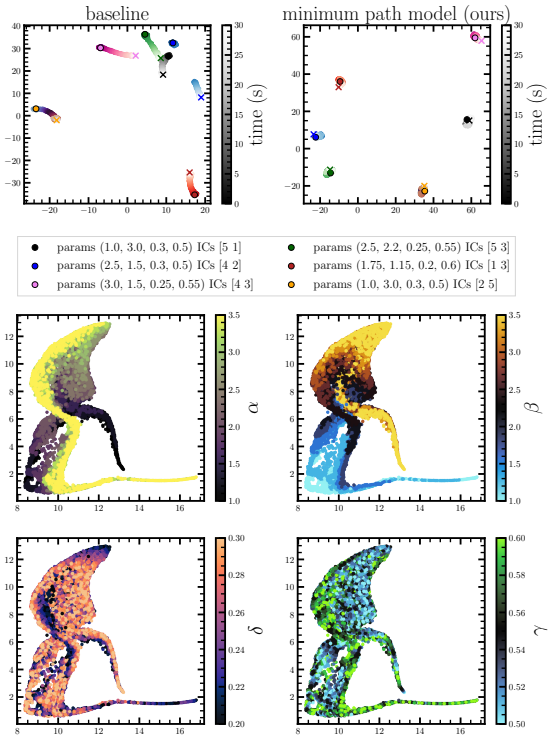

Figure 4: *Top:* UMAP projection of latent space trajectories for six different parameter and IC choices for the LVE (indicated by different colors, start and end points of the trajectories by circles and crosses). *Bottom:* UMAP projection of the latent space of our model color-coded by parameter.

We show a sample posterior plot in Figure 3 for the inference of a fixed set of parameters and ICs for a visual comparison of the model performance. The top panels show the results for our model, the bottom panels for the baseline. We plot results for inference made with 5 and 20 randomly sampled observations from the interpolation regime in gray and blue, respectively, with the true values indicated by vertical dashed lines. We can see that baseline and path-minimizing model produce latents that are inherently useful for inference. This can partially be attributed to the supervised training of the normalizing flow, which can tolerate substantial confusion of the inputs (such as the crossing of latent trajectories seen in the top panel of Figure 4) and still produce reasonable outputs. On closer inspection, we see that providing more data leads to narrower posteriors, as expected from an inference model. However, the baseline model is heavily biased and overconfident in the inference of the initial conditions in the $n = 5$ case.

We repeated these experiments by uniformly sampling both parameters and initial conditions then taking 5, 10, and 20 data points samples between $t = 0$ and $t = 15$ and reporting the relative MSE of the parameter and IC posterior means averaged over 256 trials. The results are summarized in Table 2. We test the inference capabilities for in-domain and out-of-domain parameter sets, where for the out-of-domain set we extend the

parameter bounds to exceed the training range of the latent ODE model by 50% on both the high and low ends. Our model consistently outperforms the baseline in all trials, with notably stronger performance for sparsely observed data. In fact, the posterior MSE of our model with $n = 5$ is comparable to the baseline model with $n = 20$. This behavior is retained for out-of-distribution cases.

4 Discussion and future work

Stochastic ODEs We do not test our regularization scheme on stochastic differential equations because those lie outside the scope of this paper. However, it seems reasonable to expect that an extra care is needed to model such systems. The type of stochasticity may be effectively represented in latent space [Li et al., 2020], or it may resist a low-dimensional, time-invariant representation.

Chaotic systems Another limitation of our model, and latent ODE models in general, would likely be encountered when attempting interpolation and forecasting performance for chaotic systems. While we have demonstrated good performance in coupled and non-periodic systems, finding robustness to out-of-distribution cases, the same strengths might lead to overconfident, temporally smooth predictions for systems that in reality behave chaotically. Care should therefore be taken to screen for and avoid systems in chaotic states.

State space modulation Our path-minimizing loss encourages time invariance of the latents, but because the ODE solver operates on that space, some temporal evolution needs to be allowed. We expect that the time "axis" only occupies a low-dimensional submanifold, but our latents are not fully time-independent descriptions of the system configuration. To further our goal of utilizing latent ODEs as robust inference models, we intend to combine our approach with the explicit static/dynamic state splitting of the latent parameters described by [Auzina et al., 2024].

5 Conclusion

We introduce a novel regularization approach for latent ODE models. We remove the customary variational loss and replace it with an instance-level path-length penalty in latent space. This loss function can readily be used with all existing latent ODE architectures. We adopt the latent ODE-RNN architecture and find training with the proposed loss to be more effective, reaching lower loss values with fewer iterations. The resulting models significantly improve the interpolation and extrapolation accuracy in three different test cases. We also find increased robustness to parameter choices beyond the limits of the training data.

As intended, the latent distributions show little time evolution; instead, they are mostly shaped by system parameters and initial conditions. The latent ODE encoder thus serves as an inference model, and we find accurate inference results when using the encoder to summarize irregularly sampled sequence data. Our straightforward modification to the training loss thus manifestly improves the usability of latent ODE models to empirically describe dynamic systems.

References

Ilze Amanda Auzina, Çağatay Yıldız, Sara Magliacane, Matthias Bethge, and Efstratios Gavves. Modulated neural odes. *Advances in Neural Information Processing Systems*, 36, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

C Coelho, M Fernanda P Costa, and LL Ferrás. Enhancing continuous time series modelling with a latent ode-lstm approach. *Applied Mathematics and Computation*, 475:128727, 2024.

Patrick Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, art. arXiv:1312.6114, December 2013. doi: 10.48550/arXiv.1312.6114.

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable Gradients for Stochastic Differential Equations. *arXiv e-prints*, art. arXiv:2001.01328, January 2020. doi: 10.48550/arXiv.2001.01328.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal Differential Equations for Scientific Machine Learning. *arXiv e-prints*, art. arXiv:2001.04385, January 2020. doi: 10.48550/arXiv.2001.04385.

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

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

Ruian Shi and Quaid Morris. Segmenting hybrid trajectories using latent odes. In *International Conference on Machine Learning*, pages 9569–9579. PMLR, 2021.

Daniel Ward. Flowjax: Distributions and normalizing flows in jax, 2024. URL https://github.com/danielward27/flowjax.

Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning Likelihoods with Conditional Normalizing Flows. *arXiv e-prints*, art. arXiv:1912.00042, November 2019. doi: 10.48550/arXiv.1912.00042.

Hedi Xia, Vai Suliafu, Hangjie Ji, Tan M. Nguyen, Andrea L. Bertozzi, Stanley J. Osher, and Bao Wang. Heavy Ball Neural Ordinary Differential Equations. *arXiv e-prints*, art. arXiv:2110.04840, October 2021. doi: 10.48550/arXiv.2110.04840.

A Experimental setup and ODE details

Our latent ODE-RNN architecture is implemented in `jax` and `equinox` [Bradbury et al., 2018, Kidger and Garcia, 2021]; the normalizing flow package is `flowjax` [Ward, 2024].

ODE function We use a feed forward neural network to model our ODE function f_θ as in [Rubanova et al., 2019]. We use a Tanh activation function and report the network size for all trials in Appendix B.

ODE solver For all data generation, and ODE integration of our models we use the ODE solver from the `diffrax` package [Kidger, 2021]. We use the 5^{th} order `Tsit5()` solver, Tsitouras' 5/4 method (5^{th} order Runge-Kutta), with adaptive steps and an initial $dt = 0.1$. We set the relative and absolute error tolerances of 1e-4 for both values.

A.1 Data generation

We detail the data generation steps for the test cases below. We note we perform a $80, 10, 10$ split for training/testing/validation of all datasets. We add Gaussian noise to all generated data at a level of 0.05 to encourage robustness.

Lotka-Volterra equations We generate 22,000 samples from numerical solutions to Equation 3. Each sample consists of 150 data points are randomly sampled from $t_i = 0$ to an endpoint of t_f, that we randomly choose from the $[25, 30]$. We uniformly sample for our parameters $\alpha \in [1, 3.5]$, $\beta \in [1, 3.5]$, $\gamma \in [0.5, 0.6]$, and $\delta \in [0.2, 0.3]$, and sample randomly for initial conditions between 1 and 6 for predator and pray numbers.

B Model and training details

All models are trained on a single Nvidia A100 GPU with 1 CPU and 40GB of memory.

Lotka-Volterra equations Both models presented are trained with a hidden state of dimension 16, and a latent state of dimension 8. The ODE functions consist of 3 layers of 40 units with Tanh activations. We used the Adam optimizer with a learning rate of 2e-3 and trained for up to 15,000 steps with a batch size of 64. The best trained model was chosen for both the baseline and our model. We also varied the size of the hidden dimensions, latent dimension and neural ODE size between 8 and 24, 4 and 12, 24 and 100, respectively, and found no better results for either model.

Previous studies [Shi and Morris, 2021, Auzina et al., 2024] report difficulties in training the baseline model to perform well on this problem, especially for extrapolation. They used complicated training schemes involving iterative growing scheme [Rackauckas et al., 2020], and/or sequentially increasing the size of the training time over multiple runs. We do not use any of these methods and simply perform a single training phase on a fixed training set.

We performed a large parameter sweep to find optimal values of λ finding anything in the range of $\lambda \in [0.5, 2]$ to give us the best results and all reported results are from the trial with $\lambda = 0.5$.

Effect of penalty strength The λ parameter in Equation 1 regulates the relative strength of the path minimization loss term compared to the reconstruction loss term. Optimal values for our three test cases we all within an order of magnitude of unity. To get a good first guess of λ, we try to equate early values of the reconstruction loss with the path minimization loss. This procedure requires minimal computation time and is allows for efficient hyperparameter searches.

Normalizing flow We train a normalizing flow to perform parameter inference on the predator-prey using the `flowjax` library Ward [2024]. We train a neural autoregressive flow with a dataset of 10,000 trajectories with known parameters and initial conditions which are stored in an input vector x. We vary the amount of points in the trajectory from 5 to 20 reporting results for the 5, 10, and 20 trials. We also test on out-of-distribution samples. The full training implementation of the normalizing flow involves taking the trajectories in observation space and then creating the latent context vectors u with the trained latent ODE models for each implementation. The normalizing flow is trained to predict the parameters x given u for each model. We run to 500 epochs with a learning rate of $2e-3$,

the normalizing flow has a depth of 1. We report the best results for each model in the main contents of this paper.

C Further results

C.1 Latent space structure in predator-prey system

We show additional UMAP projections of the latent space of both the baseline and our path-minimizing models for the predator-prey system in Figure 5 and Figure 6. We show the parameters with free initial conditions, meaning we randomly sample the ICs over the 10,000 trials used to create the UMAP in all figures. This allows us to see that also the ICs are clearly located in latent space. Some of the trends are a little less clear than in Figure 4 because now six variable parameters are mapped onto two UMAP dimensions, but it is evident that the latents show sharper structures in our model than in the baseline, which is the reason for the improvements in the inference results.

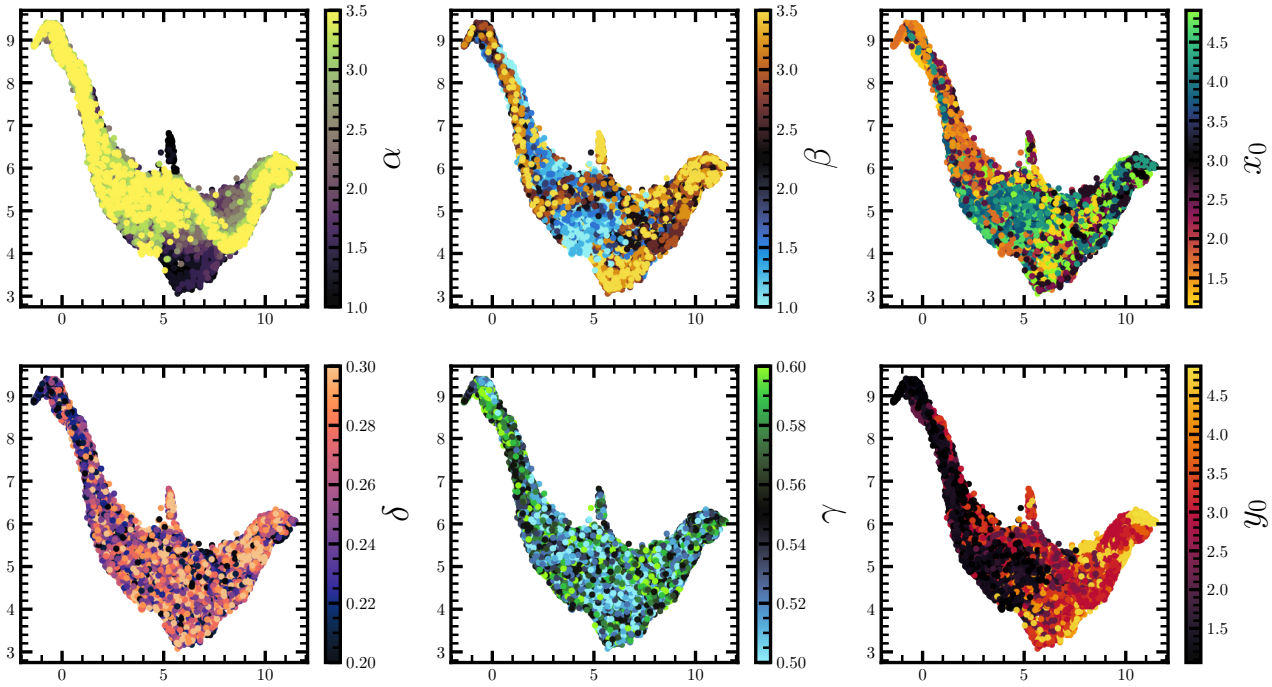

Figure 5: Same as the bottom panel of Figure 4, except we also vary the initial conditions when constructing the UMAP showing them in the two rightmost panels.

C.2 Encoder tests

We show sample results from a damped harmonic oscillator test for ODE-GRU and ODE-LSTM encoder/decoder models in Figure 7 to show our loss modification works on arbitrary encoder/decoder models.

	ODE-LSTM	ODE-LSTM (minpath)	ODE-GRU	ODE-GRU (minpath)
interp $\langle$MSE$\rangle$	0.13 ± 0.04	0.14 ± 0.08	0.19 ± 0.06	0.09 ± 0.05
extrap $\langle$MSE$\rangle$	2.42 ± 3.5	0.34 ± 0.21	9.51 ± 10	0.24 ± 0.19

Table 3: Averaged results for the alternate decoder tests. Similar to main paper we show the mean square error for both the interpolation region, and the extrapolation region for all models averaged over 50 trials.

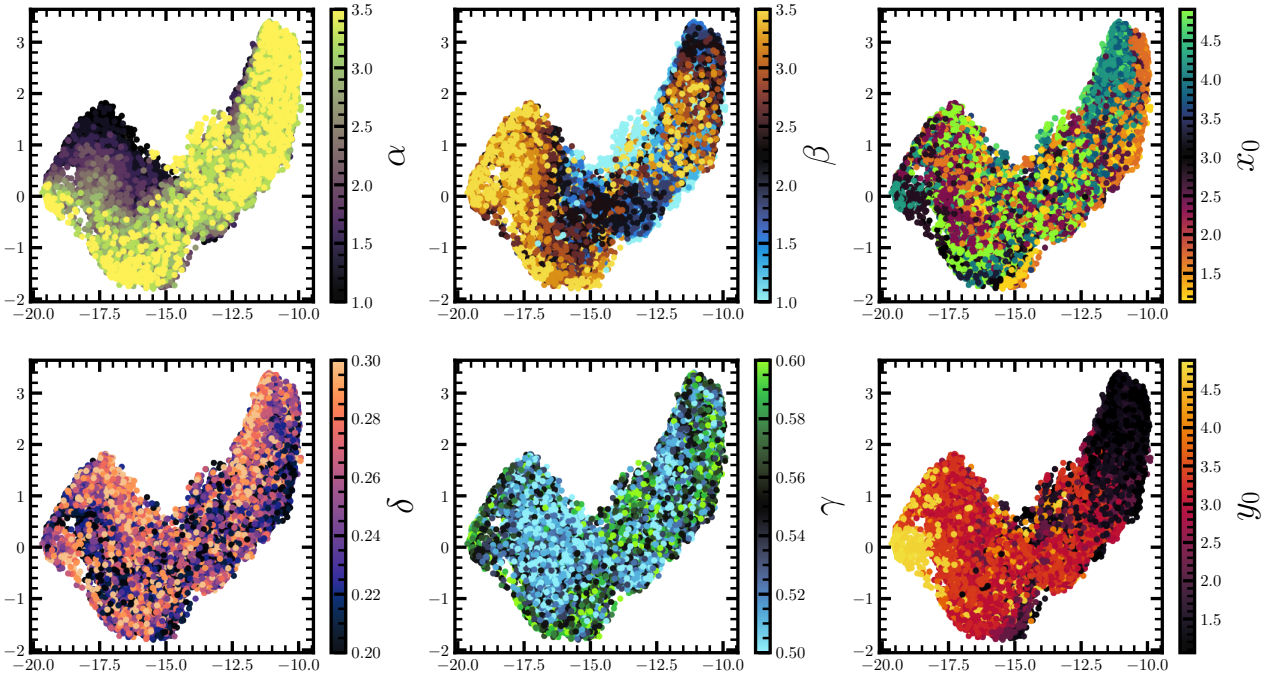

Figure 6: Same as Figure 5, but for the baseline model.

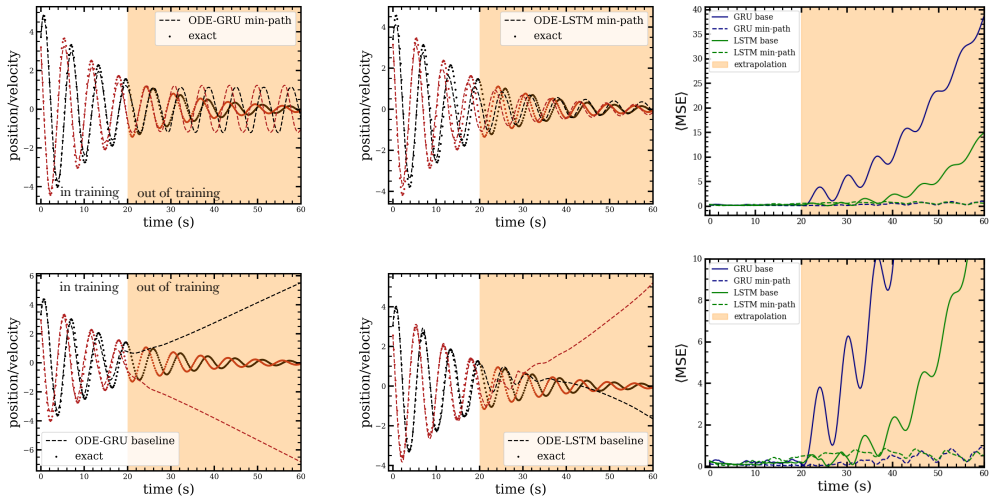

Figure 7: Tests on a simple harmonic oscillator system for GRU and LSTM encoder models in the left and middle columns respectively. We show the standard models in the bottom row and our additional loss term in the top. We see improved performance specifically in the extrapalatory regions (orange shading) in models with the path-minimisation.