A. PELT Algorithm

The pruned exact linear time (PELT) algorithm is shown below:

Algorithm 1: PELT Algorithm

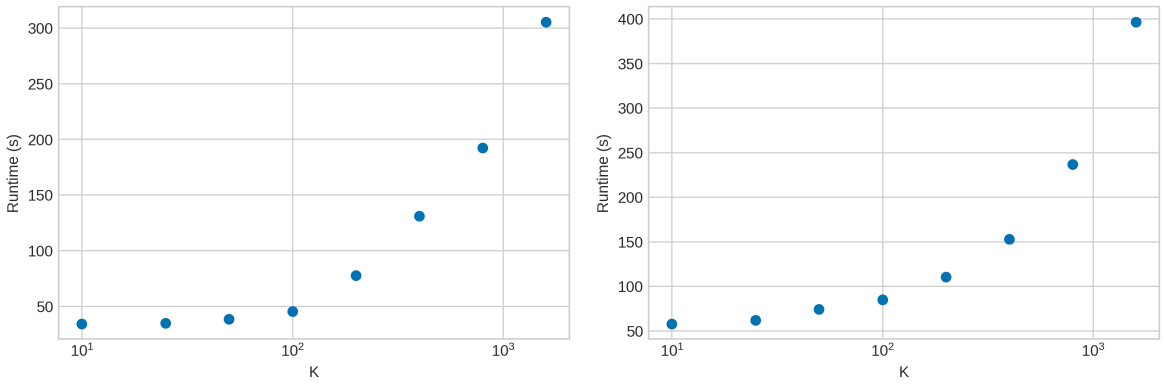
Input: Data observations $x_{1:N}$. Cost Function $\mathcal{C}$. Penalization β. Pruning parameter K.
Output: cp, position of changepoints.

1 Initialize $F(0) = -\beta$. $cp = \{\}$, $R_1 = \{0\}$
2 **for** $\tau^* = 1, ..., N$ **do**
3 Calculate $F(\tau^*) = \min_{\tau \in R_{\tau^*}} [F(\tau) + \mathcal{C}(x_{(\tau+1):\tau^*}) + \beta]$
4 Let $\tau' = \arg\min_{\tau \in R_{\tau^*}} [F(\tau) + \mathcal{C}(x_{(\tau+1):\tau^*} + \beta)$
5 Append τ' to cp
6 Set $R_{\tau^*+1} = \{\tau \in R_{\tau^*} \cup \{\tau^*\} : F(\tau) + \mathcal{C}(x_{(\tau+1):\tau^*}) + \beta \leq F(\tau^*)$
7 **return** cp

Array R stores the changepoints under consideration for the optimal segmentation. Line 6 contains the pruning condition of PELT. After computation of the optimal sub-solution, PELT removes changepoints which cannot be optimal, as determined by equation (16-17) of the main text. In LatSegODE, we flip this algorithm to perform maximization instead of minimization by switching the relevant signs. We set β to be zero, and use the marginal likelihood from equation (15) of the main text as $\mathcal{C}$. The choice of K is problem dependent. Increasing K increases the threshold required to prune a changepoint. This increases segmentation accuracy at the cost of runtime.

B. Runtime

Here, we document the trade-off between K and segmentation runtime. Setting a high K allows more changepoints to be considered for the optimal segmentation. This means an optimal changepoint will be accidentally pruned with less frequency. Naturally, considering additional changepoints increases runtime, which we visualize in Figure 1.



(a) One ground truth changepoint.

(b) Two ground truth changepoints.

Figure 1. Run time as a function of K. X axis is in log scale.

We vary the K term, and measure the wall runtime of LatSegODE segmentation. In subplot (a), we segment trajectories from the Lotka Volterra data set with one changepoint. These trajectories contains 410 observations on average. In subplot (b), we segment trajectories from the Lotka Volterra data set containing two changepoints containing 620 observations on average. We observe that segmentation accuracy increases as we increase K. This relationship is shown in Table 1. Accuracy when using low values of K is very poor. Accuracy rapidly increases as K is increased, and then plateaus. For the Lotka-Volterra benchmark, a K value around 50-100 offers the best performance for its runtime.

K	True CP = 1				True CP = 2			
	Rand Index	Hausdorff Metric	F1 Score	Annot. Error	Rand Index	Hausdorff Metric	F1 Score	Annot. Error
10	0.7781	131.6	0.5578	1.4	0.8662	150.8	0.5714	0.133
25	0.8344	103.8	0.8133	0.4	0.9342	105.8	0.6929	1.0
50	0.9024	66.4	0.9333	0.2	0.9585	24.6	0.5524	0.6
100	0.9016	66.6	0.9333	0.2	0.9659	19.6	0.5619	0.4
200	0.9031	66.2	0.9333	0.2	0.9582	22.4	0.4952	0.4

Table 1. Segmentation accuracy as a function of K. Metrics averaged over 5 runs.

C. Neural Event ODEs

Here, we demonstrate our intuition on why the Neural ODE and Neural Event ODE methods are unable to converge within our problem domains. Neural ODEs represent trajectories as a deterministic function of its initial state. Furthermore, two different trajectories cannot evolve from an identical initial state. In experimental benchmarks, and time series modelling in general, trajectories may start at identical initial states but later diverge. Neural ODEs and Neural Event ODEs, which can only encode one trajectory per initial state, are not designed to represent this class of time series. Consequently, we observe that Neural ODEs and Neural Event ODEs do not converge during training on our benchmarks. An example is shown in Figure 2.

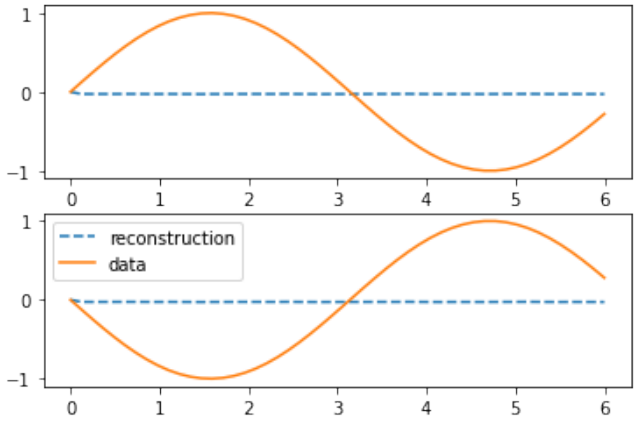


Figure 2. An example of trajectories which cannot be simultaneously represented by a single Neural Event ODE. Both trajectories begin with an initial value of zero, but evolve with opposite dynamics.

The figure shows two sine waves with opposite amplitudes a shared initial value of zero. As expected, we see that Neural Event ODE reconstructions are poor, and did not converge during training. The Neural Event ODE was trained using the sine wave benchmark data set. The ODE function is parameterized using a 3 layer neural network (NN) with Tanh activations and 256 units in hidden layers. We use a 2 layer NN with 256 width and ReLU activations to parameterize the update and event functions. Hyper-parameter search was performed by varying the number of layers and hidden layer width in the ranges (2, 3) and (64, 128, 256).

Latent ODEs does not suffer this representational limitation. We consider the example in the figure. The Latent ODE can represent this example by using two different latent trajectories which begin at non-identical latent initial states. The decoder neural network can then learn a surjective function which maps the non-identical latent initial states to identical initial values in data space. Extension of the Neural Event ODE to a latent architecture is a promising direction to circumvent limitations.

D. General Experimental Design

All experiments are run on a single Titan Xp GPU. Due to computational constraints, each training run was only performed once per hyper-parameter choice. Each segmentation evaluation was also only run once due to computational constraints.

D.1. Reconstructive Baselines

Reconstructive performance was baselined using several methods. First, we selected the GRU model, a standard choice in representing time series. The GRU Δt model uses the time delta between observations as an additional input feature. Inclusion of the time delta improved extrapolation performance. The GRU-ODE model uses a Neural ODE to evolve hidden state between GRU updates. All of these methods are trained in an auto-regressive manner. During training, scheduled sampling (Bengio et al., 2015) is used. At each time step, with 25% probability, the input data is replaced with the prediction output of the previous time point. We also mask 25% of the observations at the end of the trajectory (in addition to masking described in main text), and include the prediction loss on these points. These models are trained using the mean squared error loss between ground truth and predicted data points.

As the LatSegODE is provided changepoint positions (and thus isolated simple dynamical flows) during training, we also provide this information to baseline methods. We trained models on trajectories which have been split into simple dynamical flows (SDFs), and also separately trained on entire hybrid trajectories. The best performing model resulting from these two training configurations is reported.

We trained vanilla Latent ODEs using only whole hybrid trajectories. Obtaining convergence was difficult, and required days of training. We use the training strategy from (Rackauckas et al., 2020), where we iterative grow the learned reconstruction. We experimented using an augmented Neural ODE (Dupont et al., 2019) to represent latent dynamics, but did not observe significant benefit. We applied an adjoint computation modification to speed up training (Kidger et al., 2020), but did not perform rigorous empirical testing to evaluate its benefits.

We hacked data generation to enable faster experimentation. Generated observation times were irregularly sampled, but aligned for all trajectories. This enables batch training. In practice, Latent ODEs can handle non-aligned irregularly sampled time series by solving for the union of all time points in a mini-batch. This is drastically increases memory usage and runtime. Thus, for convenience, we adopt the aforementioned hack.

D.2. Segmentation

We report the specific formulations of the CPD methods and metrics used in our benchmarks. These descriptions are referenced from the `ruptures` (Truong et al., 2020) package documentation.

METRICS

The Rand Index (Rand, 1971) measures the overlap between the predicted segmentation and the ground truth segmentation for a segmentation S on data points $x_{1:T}$. A membership matrix A is defined such that $A_{ij} = 1$ if x_i and x_j are in the same segment. Otherwise, $A_{ij} = 0$. Membership matrices are generated for both the ground truth segmentation (A) and the predicted segmentation $(\tilde{A})$. The Rand index is defined as:

$$\text{RandIndex} = \frac{\sum_{i<j} \mathbb{1}[A == \tilde{A}]}{T(T-1)/2} \tag{1}$$

The Hausdorff metric (Rockafellar & Wets, 2009) is a measure of the maximal distance between the predicted segmentation and the ground truth segmentation. Given a set of ground truth changepoints $t_1, t_2, ...$ and predicted changepoints $\hat{t_1}, \hat{t_2}, ...$, it is computed as:

$$\text{Hausdorff}(\{t_k\}_k, \{\hat{t}_k\}_k) = \max\{\max_k \min_l |t_k - \hat{t}_l|, \max_k \min_l |t_l - \hat{t}_k|\} \tag{2}$$

Intuitively, it returns the max of the set of distances from each predicted changepoint to their closest ground truth changepoint.

The F1 score is calculated as the standard F1 score, namely the harmonic mean between precision and recall:

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

A changepoint prediction is considered correct if a falls within 10 indices of a true changepoint. This definition of correctness is used to calculate precision and recall for the F1 score.

The annotation error reports the difference between the count of predicted changepoints, and the count of true changepoints.

METHODS

We outline the three cost functions used in the main text. Other cost functions (L1/L2 deviation, rank transformation, Mahalanobis distance) were evaluated not reported due to inferior performance. These cost functions are used with PELT. Alternative search methods such as binary segmentation and sliding windows were not considered. They are prone to returning sub-optimal results, as they are greedy methods.

RPT-RBF: This cost function detects changes in the distribution of a sequence of i.i.d. random variables. It introduces a kernel $k(.,.) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and a feature map $\Phi : \mathbb{R}^d \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space. RPT-RBF embeds a signal as $\{\Phi(x_t)\}_t$, and detects changes in the mean so that the cost function on an interval I is:

$$c(x_I) = \sum_{t \in I} ||\Phi(x_t) - \bar{\mu}||^2_{\mathcal{H}} \tag{4}$$

where $\bar{\mu}$ is the empirical mean of the sub-trajectory $\{\Phi(x_t)\}_{t \in I}$. RPT-RBF uses the radial basis function, which is defined as:

$$k(x, y) = \exp(-\gamma ||x - y||^2) \tag{5}$$

where $||.||$ is the Euclidean norm and $\gamma > 0$ is a smoothing parameter known as the bandwidth. Two other kernels are provided, which use a cosine similarity kernel, and a linear model. We use the RBF as it provided better performance.

RPT-NORM: This cost function scores a segment using a sequence of multivariate Gaussian random variables, i.e., a Gaussian Process. For a signal $\{x_t\}_t$ on interval I, this cost is defined as:

$$c(x_I) = |I| \log\det \hat{\Sigma}_I \tag{6}$$

where $\hat{\Sigma}_I$ is the empirical covariance matrix of the data points $\{x_t\}_{t \in I}$.

RPT-AR: This cost function introduces an auto-regressive model. It represent unknown changepoint indices as $0 < t_1 < .. < t_N$. A piece-wise auto-regressive model is introduced:

$$x_t = z_t' \delta_j + \epsilon_t \qquad \forall t = t_j, ..., t_{j+1} - 1 \tag{7}$$

To clarify, t_j is the segmentation boundary, while t represents actual time indices. Thus, $j > 1$ represents the number of a segment. The variable $z_t = [x_{t-1}, ...x_{t-p}]$ is the lag vector, with p being the order of the auto-regressive model.

On an interval I, the cost is defined as:

$$c(x_I) = \min_{\delta \in \mathbb{R}^p} \sum_{t \in I} ||x_t - \delta' z_t||_2^2 \tag{8}$$

For this cost function, the lag term is a hyper-parameter. We chose the default value (ten) and a small grid search around this value confirmed it performs the best.

E. Sine Wave Experimental Setup

We simulate 7500 total trajectories, of which 300 is used for validation and 150 for test. Each trajectory can contain up to two changepoints. The time length and number of observations in each SDF between changepoints is uniformly sampled from the ranges $(3, 5)$ and $(50, 150)$ respectively. The amplitude and frequency of SDFs are uniformly sampled from the ranges $(-8, 8)$ and $(2, 4)$, respectively. The phase of each SDF is also randomly drawn, which introduces jump discontinuities. The time points of observation were randomly sampled using a uniform distribution on the range of the segment, but were forcibly aligned for experimental convenience. The absolute change in amplitude between SDFs is at least 2.5, and we added independent Gaussian noise with standard deviation 0.025 to simulate noise. At evaluation time, all segmentation methods use a minimum segment length of 20, and we use a K term of 200.

Next, we report architecture hyperparameters and training procedure. Unless otherwise stated, hyper-parameter search was mainly performed on by adjusting the depth and width of NNs used to parameterize the Neural ODEs in various architectures. We start with a 3 layer NN with 50 units, and increase width/depth until the model can fit the data.

GRU / GRUΔt The GRU model used 100 units in the GRU, and a 20 dimensional hidden state. A 2 layer NN with 100 units and ReLU activations is used for the output network. The GRU Δt model used an identical architecture. These

hyperparameters were selected using grid search. We started with 50 units in the GRU, and increased number of units until overfit occurred. Both models were trained using Adamax (Kingma & Ba, 2015), with a learning rate of 0.01. Learning rate was manually decayed to 1e-3 and then 1e-4 when loss plateaued. A batch size of 512 was used. This was selected based on memory constraints, as larger batch sizes allowed for faster convergence.

GRU-ODE The GRU-ODE model used 100 units in the GRU, and a 10 dimensional hidden state. The output network is a 2 layer 100 unit NN with ReLU activations. The Neural ODE component of the GRU-ODE is parameterized by a 3 layer 100 unit NN with Tanh activations. We found that using batch size 1 provided good results, at the cost of a very long training run. The model was optimized using Adamax, with manual learning rate reduction on plateaus from 0.01 to 1e-3 to 1e-4.

Latent ODE The vanilla Latent ODE model used a encoder with a 32 dimensional hidden state. The latent dynamics were 16 dimensional. The dimensionality in the vanilla Latent ODE is larger than the base model used in the LatSegODE, as the latent initial state for vanilla Latent ODEs must encode more information. The encoder GRU contains 100 units, and the encoder Neural ODE was parameterized by a 3 layer 100 units NN with Tanh activations. The Neural ODE representing latent dynamics was parameterized identically. We used a 2 layer 100 unit NN with ReLU activations as the decoder. The Latent ODE is trained using Adamax with learning rate 0.01. The learning rate was manually decayed to 1e-3 when loss plateaued. A batch size of 128 was used. We used KL annealing (Fu et al., 2019), such that the KL weight was 0 at the start of training, and reached 1 at epoch 5. The fixed variance used to compute the ELBO was set to 1. Latent dynamics were solved using the dopri5 solver using relative and absolute tolerances of 1e-8 and 1e-8. The encoder Neural ODE was solved using Euler's method.

LatSegODE The LatSegODE used a 10 dimensional hidden state in the encoder, and a 5 dimensional latent dynamical state. The original Latent ODE paper observed that the dimension of the encoder hidden state must be larger than the dimension of the latent dynamical state. Our experimentation supported this claim, and training was not possible if this rule was violated. The GRU in the GRU-ODE encoder uses 100 units, and the encoder Neural ODE is a 2 layer 100 unit NN with Tanh activations. The Neural ODE used to represent latent dynamics used the same hyperparameters. We found that Tanh activations were required for stable training in Latent ODEs. We attempted other activations such as Swish, ReLU, and Softplus, but found this caused numerical under/overflow. We used a 2 layer 100 unit decoder network with ReLU activations.

It is important to use ReLU activations in the decoder network. Alternative activations such as the Tanh or SoftPlus are bijective activations, meaning the initial observed data point in data space can never map to multiple latent initial states. Using the Tanh activation greatly limits the representational power of the Latent ODE base model when representing data trajectories which start at the same value, but later diverge.

The LatSegODE was trained using Adamax using learning rate 0.01, reduced to 1e-3 manually when validation loss plateaued. We used a batch size of 256. We used the dopri5 solver to solve latent dynamics with relative tolerance of 1e-5 and absolute tolerance of 1e-6. Increasing the ODE solve tolerance had no adverse effects on results and sped up training. The encoder Neural ODE used Euler's method to solve dynamics. We used KL annealing during training.

The fixed variance used to compute ELBO is set to 1. At segmentation time, it was critical to use the same fixed variance to evaluate marginal likelihood. We used 100 Monte Carlo samples to estimate marginal likelihood. We used a K term of 200. To reduce the memory cost of segmentation, we rounded times of observation to the nearest 0.01, reducing the size of the union of time points which must be solved.

F. Sine Wave Results

Experimental results on the Sine Wave dataset. The Latent ODE trained on input augmented with a binary time series of changepoint locations is denoted Aug. Latent ODE.

	Reconstruction			Segmentation			
Method	Total MSE	Interp. MSE	Extrap. MSE	Rand Index (↑)	F1 Score (↑)	Hausdorff Metric (↓)	Annot. Error
LatSegODE	**1.933**	**0.639**	**4.266**	**0.993**	**0.989**	**2.107**	0.033
GRU	8.065	0.862	20.634	-	-	-	-
GRU Δt	6.911	0.973	17.560	-	-	-	-
GRU-ODE	6.616	0.815	17.130	-	-	-	-
Latent ODE	12.877	9.111	17.721	-	-	-	-
Aug. Latent ODE	13.273	4.914	26.922	-	-	-	-
RPT-RBF	-	-	-	0.918	0.837	25.220	-
RPT-AR	-	-	-	0.938	0.887	26.013	-
RPT-NORM	-	-	-	0.855	0.761	47.213	-

Table 2. Evaluation on Sine Wave data set. Values are averages over 150 test trajectories. Arrows beside metric denote whether higher values (↑) or lower values (↓) indicate better performance.

G. Lotka-Volterra Experimental Setup

Here, we report the parameters used to generate the Lotka Volterra data set, and the model hyper-parameters used during bench marking. As previously reported, we simulated 34000 training hybrid trajectories, with 600 trajectories used as a validation set, and 150 used for test. Each trajectory contained zero to two changepoints. Changepoints in these trajectories are labelled, and the LatSegODE base model is trained on the SDFs between changepoints. The vanilla Latent ODE baseline is trained only on the hybrid trajectories, while other baselines were trained both on full hybrid trajectories and SDFs separately, with the best result reported.

Each SDF between changepoints was generated with between 175 to 225 observations, and ends at a time randomly sampled between $(14, 16)$. The coefficients for the Lotka-Volterra SDFs were uniformly sampled from ranges $(0.5, 1.5), (0.5, 1.5), (1.5, 2.5), (0.5, 1.5)$ for coefficients $\alpha, \beta, \delta, \gamma$ respectively. We enforced a minimum change in the norm of coefficients of 0.6 between SDFs, and added independent Gaussian noise of 0.01 to all trajectories. We masked 20% of data points for interpolation testing, and the last 100 data points for extrapolation testing, as reported in the main text.

In the next appendix, we separately evaluate performance on data sets containing jump discontinuity (JD) and without (SD). For the JD set, each SDF within a trajectory is restarted at a new initial population at changepoints, uniformly sampled from ranges $(1.5, 2.5), (0.5.1.5)$ for x, y respectively. In the SD set, SDFs are initialized from the same distribution, but remain continuous at subsequent changepoints. To clarify, trajectories in the SD set do not contain jump discontinuities at changepoints, and these locations only feature a switch in dynamical mode.

Next, we report the hyperparameters and training procedures in the Lotka Volterra experiments. We performed hyperparameter search over the width and depth of the NNs parameterizing Neural ODEs in our architectures. We started with a one layer 64 unit NN, and increased the number of layers and units until convergence was possible. Latent dimension selection was difficult. We started with a latent dimension of 64, and slowly decreased capacity until under-fitting occurred.

GRU / GRUΔt We found that the GRU always performed worse than the GRUΔt, so we did not report its results. The GRU Δt model used 16 dimensions in its hidden state, and 200 units in the GRU. A 2 layer 100 unit NN with ReLU activations was used as an output network. The model was trained using Adamax, where learning rate was decayed from 1e-2 to 1e-3 to 1e-4 each time validation loss plateaued. We used a batch size of 512.

GRU-ODE The GRU-ODE model used a hidden state with dimension 50, and GRU with 100 units. A 2 layer 100 unit output network with ReLU activations was used. The Neural ODE was parameterized with using a 3 layer 200 unit NN with Tanh activations. The GRU-ODE was trained by Adamax using a constant 1e-3 learning rate. The Neural ODE solutions were solved using dopri5 with relative and absolute tolerances of 1e-3 and 1e-4.

Latent ODE Latent ODE models were very hard to train, and many approaches were taken. First, we attempted to tune hyperparameters, increasing both latent dimension size, and the number of parameters in NNs parameterizing Neural ODEs. The rationale was that the added complexity in hybrid trajectories necessitated increased model complexity. We found that model reconstruction accuracy in both data sets were poor even after many epochs. A more successful strategy was to adopt an iterative growing scheme for training (Rackauckas et al., 2020). We also adopted a forward prediction training strategy. We started training on a masked sub-trajectory with only the first 50 data points, and evaluate loss on prediction of the next 50 data points. The training loss was computed using all observed points. After each epoch, we increased the number of observed data points by 50. We also found that taking multiple samples of the latent initial state during training could stabilize gradient estimates, and we used 3 samples per trajectory.

Latent ODEs used were trained using Adamax with a constant learning rate of 5e-3, decayed to 1e-3 after 10 epochs. A batch size of 256 is used. Latent ODEs were trained using KL annealing such that a KL weight of 1 was reached after 10 epochs. A fixed variance of 0.01 was used in the ELBO.

The Latent ODE models had a hidden state of dimension 16, and a latent state of dimension 8. The encoder used 100 GRU units, with its Neural ODE parameterized by a 3 layer 100 unit NN with Tanh activations. The identical hyper parameters are used for the latent Neural ODE neural network. We used a linear decoder. Latent dynamics were solved using dopri5 with relative and absolute tolerance of 1e-4 and 1e-4.

LatSegODE The LatSegODE used a base model with a hidden state of dimension 16, and a latent state with dimension 8. The encoder used GRUs with 100 units, and both the encoder Neural ODE and latent Neural ODE was parameterized by 3 layer 100 unit NNs with Tanh activations. A decoder network with 2 layer 100 units with ReLU activations was used.

We trained our model using Adamax, with an initial learning of 5e-3. Learning rate was decayed to 1e-3 after 10 epochs, and reduced to 1e-4 after validation loss plateaued. A batch size of 256 was used. Latent dynamics were solved using dopri5, with relative and absolute tolerances of 1e-4 and 1e-4. Training used KL annealing such that KL weight started from 0 and reached 1 after 10 epochs. A fixed variance of 0.01 was used to compute the ELBO.

At segmentation time, we used a fixed variance of 0.01 to compute the marginal likelihood. We took 100 MC samples to compute the marginal likelihood, and rounded times of observation to 2 decimal places.

H. Lotka-Volterra Expanded Experiments

We report the full results of Lotka-Volterra experiments on additional data sets, jump discontinuous (JD) and switching dynamics (SD). The JD set is identical to the set reported in the main text. The SD does not contain jump discontinuity at changepoints, and the trajectory only switches to a new dynamical mode. The SD set is included evaluate performance of the LatSegODE without discontinuous jumps, which may be theoretically easier to solve since ODE solves do not need to bridge a jump discontinuity. We find the opposite actually occurs, as the less distinct changepoints in the SD set yields a harder data set, shown by the decreased performance of baselines. We report the metrics in Table 3 below.

	Reconstruction			Segmentation			
Method	Total MSE	Interp. MSE	Extrap. MSE	Rand Index (↑)	F1 Score (↑)	Hausdorff Metric (↓)	Annot. Error
LatSegODE	**0.068**	0.0312	**0.2396**	**0.9464**	**0.8268**	**47.67**	0.76
GRU Δt	0.1718	**0.0193**	0.8329	-	-	-	-
GRU-ODE	0.2747	0.1201	2.0358	-	-	-	-
Latent ODE	0.6155	0.5072	0.9505	-	-		-
RPT-RBF	-	-	-	0.7956	0.4167	84.7	-
RPT-AR	-	-	-	0.6994	0.4367	164.65	-
RPT-GPNorm	-	-	-	0.7693	0.42	105.92	-

(a) Jump discontinuous (JD) test set.

	Reconstruction			Segmentation			
Method	Total MSE	Interp. MSE	Extrap. MSE	Rand Index (↑)	F1 Score (↑)	Hausdorff Metric (↓)	Annot. Error
LatSegODE	**0.2284**	0.2176	**1.010**	**0.8758**	**0.6857**	**75.66**	2.42
GRU Δt	0.4519	**0.0383**	2.427	-	-	-	-
GRU-ODE	0.4386	0.2176	3.280	-	-	-	-
Latent ODE	1.4027	1.1924	2.1314	-	-	-	-
RPT-RBF	-	-	-	0.7833	0.4233	77.77	-
RPT-AR	-	-	-	0.7020	0.4433	138.24	-
RPT-GPNorm	-	-	-	0.7672	0.4233	94.44	-

(b) Switching dynamical mode (SD) test set.

Table 3. Expanded results on Lotka Volterra hybrid trajectory benchmark. Values are averages over 150 test trajectories. Arrows beside metric denote whether higher values (↑) or lower values (↓) indicate better performance.

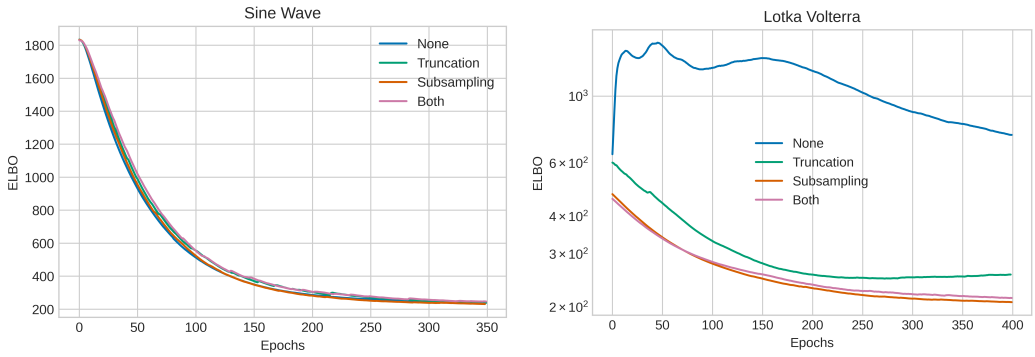
I. Data Augmentation Strategies

We report two data augmentation techniques which can increase training speed and the ability for the Latent ODE to generalize. These techniques are applied on data batches prior to each training iteration. First, we propose sub-sampling trajectories by randomly removing a percentage of observed points. This method resembles techniques which iteratively grows trajectory length throughout training to avoid local minima (Rackauckas et al., 2020), and we hypothesize sub-sampling confers a similar benefit. We also introduce start-truncation, where the first N data observations are cropped. We hypothesize this augmentation decreases generalization error by exposing the Latent ODE encoder to more initial states.

To demonstrate the efficacy of these augmentation methods, we generate 10000 SDFs from each of the previous experimental domains, with 500 trajectories held for validation and 500 for test evaluation. Each trajectory contains 200 samples. For sub-sampling, we randomly select between 40 to 200 data points to train per batch. For truncation, we randomly select the number of data points to remove from range $(0, 160)$. We train Latent ODE models with and without augmentation techniques, and plot the validation loss curve for each training run in Figure 3.

The augmentation methods accelerate training, and achieve faster convergence. In Table 4, we report the generalization error calculated using test trajectories, and the wall time for training to complete 350 / 400 epochs, which both decrease. We report parameters used to run the data augmentation experiments.

The Sine Wave data set was generated using amplitudes and frequencies sampled from $(-8, 8)$ and $(2, 4)$ respectively. Phase was randomly sampled. 20000 training trajectories of length 5 and with 200 samples were generated. 2000



(a) (b)

Figure 3. Validation loss curves using data augmentations. Plot (a) shows runs on Sine Wave data, while (b) shows runs on Lotka Volterra data.

	Sine Wave		Lotka Volterra	
Method	Test MSE	Runtime (hrs)	Test MSE	Runtime (hrs)
None	1.8032	13:10	3.9742	22:55
Subsampling	0.1750	9:06	0.1005	18:44
Truncation	0.14713	9:28	0.6525	18:21
Both	0.3405	6:18	0.7672	13:11

Table 4. Comparison of augmentation methods.

validation trajectories were generated, and 2000 test trajectories were generated. The Lotka-Volterra data set was generated using coefficients sampled from range $(0.5, 1.5), (0.5, 1.5), (1.5, 2.5), (0.5, 1.5)$ for $\alpha, \beta, \delta, \gamma$. The initial populations were sampled from range $(1.5, 2.5), (0.5, 1.5)$. 10000 training trajectories were generated, with 500 validation trajectories and 500 test trajectories. Training trajectories contained 200 samples, while validation and test trajectories contained 100 samples. Trajectories were all of length 15.

The Latent ODE in both Lotka Volterra and Sine Wave training runs were trained using Adamax with constant 1e-3 learning rate. A batch size of 256 was used. A latent dimension of 8 was used, while the encoder hidden state was of dimension 200. GRUs contained 200 units. A decoder network with 2 layer and 50 units with ReLU activation was used. In the Lotka Volterra data set, a 3 layer 200 units NN with Tanh activations parameterized both the encoder and latent Neural ODEs. In the Sine wave data set, the neural network width was reduced to 100.

J. Character Trajectory Experimental Hyperparameters

The LatSegODE base model was trained using a latent dimension of 8, and an encoder hidden state dimension of 16. Encoder GRUs contained 200 units. We parameterized the Neural ODEs in the encoder and latent dynamics using a 5 layer 200 unit NN with Tanh activations. The decoder network was a 3 layer 200 unit NN with ReLU activations. Hyperparameter search was performed by modifying the number of units in hidden layers and number of layers in all Neural ODE NNs. We started with 128 units per layer and 2 layers, and gradually increased the number of parameters until a good reconstruction was found.

The base model was trained using Adamax at a learning rate of 1e-3, reduced to 1e-4 when validation loss plateaued. Data augmentation was used, randomly sampling truncation bounds from (30, trajectory length) and the number of points to sub-sample from (30, trajectory length). KL annealing was used, such that the training run started with KL weight of 0, and reached a weight of 1 at epoch 50. We clipped gradients to a norm of 2. A fixed variance of 0.01 is used to compute the ELBO. Latent dynamics were solved using the dopri5 solver, with relative and absolute tolerances of 1e-5 and 1e-5. The encoder Neural ODE dynamics were solved using Euler's method.

At segmentation time, the LatSegODE used a fixed variance of 0.01 to compute marginal likelihood using 200 MC samples. A K term of 100 was used. Time of observation was rounded to 2 decimal places. We set the minimum possible segment

length to 20, similar to baselines.

K. Sine Wave Ablation Study

We report the effects of the number of training trajectories and number of samples per trajectory on LatSegODE performance. A unique LatSegODE is trained for each combination of the two parameters. We measure segmentation performance using the previously established metrics. The Sine Wave dataset is used with the same data generation and model hyper-parameters as in the main body Sine Wave experiments. The test set contains 75 trajectories each with zero to two changepoints. Each test trajectory contains 100 observed samples. The results are shown in Table 5. See Table 2 for baseline results.

	# Trajectory Samples								
	50			100			200		
# Train Trajectory	Rand Index	F1 Score	Hausdorff Metric	Rand Index	F1 Score	Hausdorff Metric	Rand Index	F1 Score	Hausdorff Metric
3000	0.700	0.577	56.0	0.71	0.666	50.21	0.853	0.512	29.28
10000	0.663	0.601	49.6	0.842	0.813	28.2	0.968	0.961	6.29
30000	0.866	0.809	25.28	0.911	0.900	15.23	0.981	0.979	4.92

Table 5. LatSegODE performance measured by segmentation metrics, as a function of training set size and number of samples in training trajectories.

References

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1171–1179, 2015.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural ODEs. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 3134–3144, 2019.

Fu, H., Li, C., Liu, X., Gao, J., Celikyilmaz, A., and Carin, L. Cyclical annealing schedule: A simple approach to mitigating KL vanishing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 240–250, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1021.

Kidger, P., Chen, R. T., and Lyons, T. "Hey, that's not an ODE": Faster ODE adjoints with 12 lines of code. *arXiv preprint arXiv:2009.09457*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., and Ramadhan, A. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66 (336):846–850, 1971.

Rockafellar, R. T. and Wets, R. J.-B. *Variational analysis*, volume 317. Springer Science & Business Media, 2009.

Truong, C., Oudre, L., and Vayatis, N. Selective review of offline change point detection methods. *Signal Processing*, 167: 107299, 2020.