Neural Continuous-Discrete State Space Models for Irregularly-Sampled Time Series

Abdul Fatir Ansari [1 2 †] Alvin Heng [2] Andre Lim [2] Harold Soh [2 3]

Abstract

Learning accurate predictive models of real-world dynamic phenomena (e.g., climate, biological) remains a challenging task. One key issue is that the data generated by both natural and artificial processes often comprise time series that are irregularly sampled and/or contain missing observations. In this work, we propose the Neural Continuous-Discrete State Space Model (NCDSSM) for continuous-time modeling of time series through discrete-time observations. NCDSSM employs auxiliary variables to disentangle recognition from dynamics, thus requiring amortized inference only for the auxiliary variables. Leveraging techniques from continuous-discrete filtering theory, we demonstrate how to perform accurate Bayesian inference for the dynamic states. We propose three flexible parameterizations of the latent dynamics and an efficient training objective that marginalizes the dynamic states during inference. Empirical results on multiple benchmark datasets across various domains show improved imputation and forecasting performance of NCDSSM over existing models.

1. Introduction

State space models (SSMs) provide an elegant framework for modeling time series data. Combinations of SSMs with neural networks have proven effective for various time series tasks such as segmentation, imputation, and forecasting (Krishnan et al., 2015; Fraccaro et al., 2017; Rangapuram et al., 2018; Kurle et al., 2020; Ansari et al., 2021). However, most existing models are limited to the discrete time (i.e., uniformly sampled) setting, whereas data from various physical and industrial systems in the real world are sometimes only available at irregular (often sparse) intervals. Such systems are best modeled as continuous-time latent processes with irregularly-sampled discrete-time observations. Desirable features of such a time series model include modeling of stochasticity (uncertainty) in the system, and efficient and accurate inference of the system state from potentially high-dimensional observations (e.g., video frames).

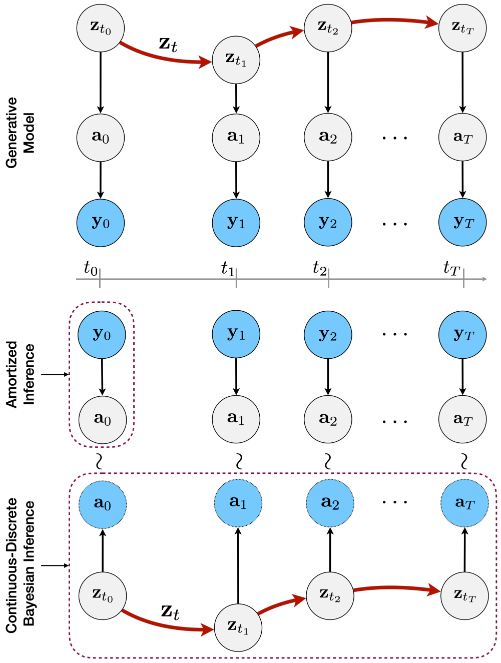


Figure 1. (**Top**) Generative model of Neural Continuous-Discrete State Space Model. The bold red arrows indicate that the state, $\mathbf{z}_t$, evolves continuously in time. The auxiliary variables, $\mathbf{a}_k$, and observations, $\mathbf{y}_k$, are emitted at arbitrary discrete timesteps $t_k \in \{t_0, t_1, \dots, t_T\}$. (**Bottom**) Amortized inference for auxiliary variables and continuous-discrete Bayesian inference for states. Samples from the amortized variational distribution over auxiliary variables are used as pseudo-observations to condition and perform inference in the continuous-discrete SSM at the bottom.

Recently, latent variable models based on neural differential equations have gained popularity for continuous-time modeling of time series (Chen et al., 2018; Rubanova et al., 2019; Yildiz et al., 2019; Li et al., 2020; Liu et al., 2020; Solin et al., 2021). However, these models suffer from limitations. The ordinary differential equation (ODE)-based models employ deterministic latent dynamics and/or encode

[†] Work done while at National University of Singapore, prior to joining Amazon. [1] AWS AI Labs [2] School of Computing, National University of Singapore [3] Smart Systems Institute. Correspondence to: Abdul Fatir Ansari <abdulfatir@u.nus.edu>.

Preprint.

the entire context window into an initial state, creating a restrictive bottleneck. On the other hand, stochastic differential equation (SDE)-based models use stochastic latent dynamics, but typically perform a variational approximation of the latent trajectories via posterior SDEs. The posterior SDEs incorporate new observations in an ad-hoc manner, potentially resulting in a disparity between the posterior and generative transition dynamics, and a non-Markovian state space.

To address these issues, we propose the Neural Continuous-Discrete State Space Model (NCDSSM) that uses discrete-time observations to model continuous-time stochastic Markovian dynamics (Fig. 1). By using auxiliary variables, NCDSSM disentangles recognition of high-dimensional observations from dynamics (encoded by the state) (Fraccaro et al., 2017; Kurle et al., 2020). We leverage the rich literature on continuous-discrete filtering theory (Jazwinski, 1970), which has remained relatively unexplored in the modern deep learning context. Our proposed inference algorithm only performs amortized variational inference for the auxiliary variables since they enable classic continuous-discrete Bayesian inference (Jazwinski, 1970) for the states, using only the generative model. This obviates the need for posterior SDEs and allows incorporation of new observations via a principled Bayesian update, resulting in accurate state estimation. As a result, NCDSSM enables online prediction and naturally provides state uncertainty estimates. We propose three dynamics parameterizations for NCDSSM (linear time-invariant, non-linear and locally-linear) and a training objective that can be easily computed during inference.

We evaluated NCDSSM on imputation and forecasting tasks on multiple benchmark datasets. Our experiments demonstrate that NCDSSM accurately captures the underlying dynamics of the time series and extrapolates it consistently beyond the training context, significantly outperforming baseline models. From a practical perspective, we found that NCDSSM is less sensitive to random initializations and requires fewer parameters than the baselines.

In summary, the key contributions of this work are:

- NCDSSM, a continuous-discrete SSM with auxiliary variables for continuous-time modeling of irregularly-sampled (high dimensional) time series;
- An accurate inference algorithm that performs amortized inference for auxiliary variables and classic Bayesian inference for the dynamic states;
- An efficient learning algorithm and its stable implementation using square root factors;
- Experiments on multiple benchmark datasets demonstrating that NCDSSM learns accurate models of the underlying dynamics and extrapolates it consistently into the future.

2. Approximate Continuous-Discrete Inference

We begin with a review of approximate continuous-discrete Bayesian filtering and smoothing, inference techniques employed by our proposed model. Consider the following Itô SDE,

$$d\mathbf{z}_t = \mathbf{f}(\mathbf{z}_t, t)dt + \mathbf{G}(\mathbf{z}_t, t)d\mathbf{B}_t, \tag{1}$$

where $\mathbf{z}_t \in \mathbb{R}^m$ is the state, $\mathbf{B}_t \in \mathbb{R}^m$ denotes a Brownian motion with diffusion matrix $\mathbf{Q}$, $\mathbf{f}(\cdot, t) : \mathbb{R}^m \to \mathbb{R}^m$ is the drift function and $\mathbf{G}(\cdot, t) : \mathbb{R}^m \to \mathbb{R}^{m\times m}$ is the diffusion function. The initial density of the state, $p(\mathbf{z}_0)$, is assumed to be known and independent of the Brownian motion, $\mathbf{B}_t$. The evolution of the marginal density of the state, $p_t(\mathbf{z}_t)$, is governed by the Fokker-Plank-Kolmogorov (FPK) equation (Jazwinski, 1970, Ch. 4),

$$\frac{\partial p_t(\mathbf{z}_t)}{\partial t} = \mathscr{L}^* p_t, \tag{2}$$

where $\mathscr{L}^*$ is the forward diffusion operator given by

$$\mathscr{L}^*\varphi = -\sum_{i=1}^{d} \frac{\partial}{\partial x_i}\left[\varphi f_i\right] + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\left[\varphi(\mathbf{G}\mathbf{Q}\mathbf{G}^\top)_{ij}\right].$$

In practice, we only have access to noisy transformations (called measurements or observations), $\mathbf{y}_k \in \mathbb{R}^d$, of the state, $\mathbf{z}_t$, at discrete timesteps $t_k \in \{t_0, \ldots, t_T\}$. The *continuous-discrete state space model* (Jazwinski, 1970, Ch. 6) is an elegant framework for modeling such time series.

Definition 2.1 (Continuous-Discrete State Space Model)**.** A continuous-discrete state space model is one where the latent state, $\mathbf{z}_t$, follows the continuous-time dynamics governed by Eq. (1) and the measurement, $\mathbf{y}_k$, at time t_k is obtained from the measurement model $p(\mathbf{y}_k|\mathbf{z}_{t_k})$.

In this work, we consider linear Gaussian measurement models, $\mathbf{y}_k \sim \mathcal{N}(\mathbf{y}_k; \mathbf{H}\mathbf{z}_{t_k}, \mathbf{R})$, where $\mathbf{H} \in \mathbb{R}^{d\times m}$ is the measurement matrix and $\mathbf{R} \succeq 0 \in \mathbb{R}^{d\times d}$ is the covariance matrix. Given observations $\mathcal{Y}_\tau = \{\mathbf{y}_k : t_k \leq \tau\}$, we are interested in answering two types of inference queries: the posterior distribution of the state, $\mathbf{z}_t$, conditioned on observations up to time t, $p_t(\mathbf{z}_t|\mathcal{Y}_t)$, and the posterior distribution of the state, $\mathbf{z}_t$, conditioned on all available observations, $p_t(\mathbf{z}_t|\mathcal{Y}_T)$. These are known as the *filtering* and *smoothing* problems, respectively.

The filtering density, $p_t(\mathbf{z}_t|\mathcal{Y}_t)$, satisfies the FPK equation (Eq. 2) for $t \in [t_k, t_{k+1})$ between observations, with the initial condition $p_t(\mathbf{z}_t|\mathcal{Y}_{t_k})$ at time t_k. Observations can be incorporated via a Bayesian update,

$$p_t(\mathbf{z}_{t_k}|\mathcal{Y}_{t_k}) = \frac{p(\mathbf{y}_{t_k}|\mathbf{z}_{t_k})p(\mathbf{z}_{t_k}|\mathcal{Y}_{t_{k-1}})}{p(\mathbf{y}_k|\mathcal{Y}_{t_{k-1}})}. \tag{3}$$

The smoothing density satisfies a backward partial differential equation related to the FPK equation. We refer the reader to Anderson (1972) and Särkkä & Solin (2019, Ch. 10) for details and discuss a practical approximate filtering procedure in the following (cf. Appendix B.1 for smoothing).

2.1. Continuous-Discrete Bayesian Filtering

Solving Eq. (2) for arbitrary $\mathbf{f}$ and $\mathbf{G}$ is intractable; hence, several approximations have been considered in the literature (Särkkä & Solin, 2019, Ch. 9). The Gaussian assumed density approximation uses a Gaussian approximation,

$$p_t(\mathbf{z}_t) \approx \mathcal{N}(\mathbf{z}_t; \mathbf{m}_t, \mathbf{P}_t), \tag{4}$$

for the solution to the FPK equation, characterized by the time-varying mean, $\mathbf{m}_t$, and covariance matrix, $\mathbf{P}_t$. Further, linearization of the drift $\mathbf{f}$ via Taylor expansion results in the following ODEs that govern the evolution of the mean and covariance matrix,

$$\frac{d\mathbf{m}_t}{dt} = \mathbf{f}(\mathbf{m}_t, t), \tag{5a}$$

$$\frac{d\mathbf{P}_t}{dt} = \mathbf{F}_{\mathbf{z}}(\mathbf{m}_t, t)\mathbf{P}_t + \mathbf{P}_t\mathbf{F}_{\mathbf{z}}^\top(\mathbf{m}_t, t) + \mathbf{D}(\mathbf{m}_t, t), \tag{5b}$$

where $\mathbf{F}_{\mathbf{z}}(\mathbf{m}_t, t)$ is the Jacobian of $\mathbf{f}(\mathbf{z}, t)$ with respect to $\mathbf{z}$ at $\mathbf{m}_t$ and $\mathbf{D}(\cdot, t) = \mathbf{G}(\cdot, t)\mathbf{Q}\mathbf{G}^\top(\cdot, t)$. Thus, for $t \in [t_k, t_{k+1})$ between observations, the filter distribution $p_t(\mathbf{z}_t|\mathcal{Y}_t)$ can be approximated as a Gaussian with mean and covariance matrix given by solving Eq. (5), with initial conditions $\mathbf{m}_{t_k}$ and $\mathbf{P}_{t_k}$ at time t_k. This is known as the *prediction step*.

The Gaussian assumed density approximation of $p(\mathbf{z}_{t_k}|\mathcal{Y}_{t_{k-1}})$ described above makes the Bayesian update in Eq. (3) analytically tractable as $p(\mathbf{y}_{t_k}|\mathbf{z}_{t_k})$ is also a Gaussian distribution with mean $\mathbf{H}\mathbf{z}_k$ and covariance matrix $\mathbf{R}$. The parameters, $\mathbf{m}_k$ and $\mathbf{P}_k$, of the Gaussian approximation of $p_t(\mathbf{z}_{t_k}|\mathcal{Y}_{t_k})$ are then given by,

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_k^-\mathbf{H}^\top + \mathbf{R}, \tag{6a}$$

$$\mathbf{K}_k = \mathbf{P}_k^-\mathbf{H}^\top\mathbf{S}_k^{-1}, \tag{6b}$$

$$\mathbf{m}_k = \mathbf{m}_k^- + \mathbf{K}_k\left(\mathbf{y}_k - \mathbf{H}\mathbf{m}_k^-\right), \tag{6c}$$

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top, \tag{6d}$$

where $\mathbf{m}_k^-$ and $\mathbf{P}_k^-$ are the parameters of $p_t(\mathbf{z}_{t_k}|\mathcal{Y}_{t_{k-1}})$ given by the prediction step. Eq. (6) constitutes the *update step* which is exactly the same as the update step in the Kalman filter for discrete-time linear Gaussian SSMs. The continuous-time prediction step together with the discrete-time update step is sometimes also referred to as the hybrid Kalman filter. As a byproduct, the update step also provides the conditional likelihood terms, $p(\mathbf{y}_k|\mathcal{Y}_{t_{k-1}}) = \mathcal{N}(\mathbf{y}_k; \mathbf{H}\mathbf{m}_k^-, \mathbf{S}_k)$, which can be combined to give the likelihood of the observed sequence, $p(\mathcal{Y}_{t_T}) = p(\mathbf{y}_0)\prod_{k=1}^{T} p(\mathbf{y}_k|\mathcal{Y}_{t_{k-1}})$.

3. Neural Continuous-Discrete State Space Models

In this section, we describe our proposed model: Neural Continuous-Discrete State Space Model (NCDSSM). We begin by formulating NCDSSM as a continuous-discrete SSM with auxiliary variables that serve as succinct representations of high-dimensional observations. We then discuss how to perform efficient inference along with parameter learning and a stable implementation for NCDSSM.

3.1. Model Formulation

NCDSSM is a continuous-discrete SSM in which the latent state, $\mathbf{z}_t \in \mathbb{R}^m$, evolves in continuous time, emitting linear-Gaussian auxiliary variables, $\mathbf{a}_t \in \mathbb{R}^h$, which in turn emit observations, $\mathbf{y}_t \in \mathbb{R}^d$. Thus, NCDSSM possesses two types of latent variables: (a) the states that encode the hidden dynamics, and (b) the auxiliary variables that can be viewed as succinct representations of the observations. The inclusion of auxiliary variables offers two benefits; (i) it allows disentangling representation learning (or recognition) from dynamics (encoded by $\mathbf{z}_t$) and (ii) it enables the use of arbitrary decoders to model the conditional distribution $p(\mathbf{y}_t|\mathbf{a}_t)$. We discuss this further in Section 3.2.

Consider the case when we have observations available at discrete timesteps $t_0, \ldots, t_T$. Following the graphical model in Fig. 1, the joint distribution over the states $\mathbf{z}_{0:T}$, the auxiliary variables $\mathbf{a}_{0:T}$, and the observations $\mathbf{y}_{0:T}$ factorises as

$$p_\theta(\mathbf{z}_{0:T}, \mathbf{a}_{0:T}, \mathbf{y}_{0:T}) = \prod_{k=0}^{T} p(\mathbf{y}_k|\mathbf{a}_k)p(\mathbf{a}_k|\mathbf{z}_k)p(\mathbf{z}_k|\mathbf{z}_{k-1}),$$

where $\mathbf{x}_{0:T}$ denotes the set $\{\mathbf{x}_{t_0}, \ldots, \mathbf{x}_{t_T}\}$ and $p(\mathbf{z}_0|\mathbf{z}_{-1}) = p(\mathbf{z}_0)$. We model the initial (prior) distribution of the states as a multivariate Gaussian distribution,

$$p(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0; \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{7}$$

where $\boldsymbol{\mu}_0 \in \mathbb{R}^m$ and $\boldsymbol{\Sigma}_0 \succeq 0 \in \mathbb{R}^{m\times m}$ are the mean and covariance matrix, respectively. The transition distribution of the states, $p(\mathbf{z}_k|\mathbf{z}_{k-1})$, follows the dynamics governed by the SDE in Eq. (1). The conditional emission distributions of the auxiliary variables and observations are modeled as multivariate Gaussian distributions given by,

$$p(\mathbf{a}_k|\mathbf{z}_k) = \mathcal{N}(\mathbf{a}_k; \mathbf{H}\mathbf{z}_k, \mathbf{R}), \tag{8}$$

$$p(\mathbf{y}_k|\mathbf{a}_k) = \mathcal{N}(\mathbf{y}_k; f^\mu(\mathbf{a}_k), f^\Sigma(\mathbf{a}_k)), \tag{9}$$

where $\mathbf{H} \in \mathbb{R}^{h\times m}$, $\mathbf{R} \succeq 0 \in \mathbb{R}^{h\times h}$, and f^μ and f^Σ are functions parameterized by neural networks that output the mean and the covariance matrix of the distribution, respectively. We use θ to denote the parameters of the generative model, including SSM parameters

$\{\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0, \mathbf{f}, \mathbf{Q}, \mathbf{G}, \mathbf{H}, \mathbf{R}\}$ and observation emission distribution parameters $\{f^\mu, f^\Sigma\}$.

We propose three variants of NCDSSM, depending on the parameterization of $\mathbf{f}$ and $\mathbf{G}$ functions in Eq. (1) that govern the dynamics of the state:

Linear time-invariant dynamics is obtained by parameterizing $\mathbf{f}$ and $\mathbf{G}$ as

$$\mathbf{f}(\mathbf{z}_t, t) = \mathbf{F}\mathbf{z}_t \quad \text{and} \quad \mathbf{G}(\mathbf{z}, t) = \mathbf{I}, \tag{10}$$

respectively, where $\mathbf{F} \in \mathbb{R}^{m \times m}$ is a Markov transition matrix and $\mathbf{I}$ is the m-dimensional identity matrix. In this case, Eqs. (4) and (5) become exact and the ODEs in Eq. (5) can be solved analytically using matrix exponentials (cf. Appendix B.2). Unfortunately, the restriction of linear dynamics is limiting for practical applications. We denote this linear time-invariant variant as NCDSSM-LTI.

Non-linear dynamics is obtained by parameterizing $\mathbf{f}$ and $\mathbf{G}$ using neural networks. With sufficiently powerful neural networks, this parameterization is flexible enough to model arbitrary non-linear dynamics. However, the neural networks need to be carefully regularized (cf. Appendix B.3) to ensure optimization and inference stability. Inference in this variant also requires computation of the Jacobian of a neural network for solving Eq. (5). We denote this non-linear variant as NCDSSM-NL.

Locally-linear dynamics is obtained by parameterizing $\mathbf{f}$ and $\mathbf{G}$ as

$$\mathbf{f}(\mathbf{z}_t, t) = \mathbf{F}(\mathbf{z}_t)\mathbf{z}_t \quad \text{and} \quad \mathbf{G}(\mathbf{z}, t) = \mathbf{I}, \tag{11}$$

respectively, where the matrix $\mathbf{F}(\mathbf{z}_t) \in \mathbb{R}^{m \times m}$ is given by a convex combination of K base matrices $\{\mathbf{F}^{(j)}\}_{j=1}^{K}$,

$$\mathbf{F}(\mathbf{z}_t) = \sum_{j=1}^{K} \alpha^{(j)}(\mathbf{z}_t)\mathbf{F}^{(j)}, \tag{12}$$

and the combination weights, $\alpha(\mathbf{z}_t)$, are given by

$$\alpha(\mathbf{z}_t) = \text{softmax}(g(\mathbf{z}_t)), \tag{13}$$

where g is a neural network. Such parameterizations smoothly interpolate between linear SSMs and can be viewed as "soft" switching SSMs. Locally-linear dynamics has previously been used for discrete-time SSMs (Karl et al., 2016; Klushyn et al., 2021); we extend it to the continuous time setting by evaluating Eq. (12) continuously in time. Unlike non-linear dynamics, this parameterization does not require careful regularization and its flexibility can be controlled by choosing the number of base matrices, K. Furthermore, the Jacobian of $\mathbf{f}$ in Eq. (5) can be approximated as $\mathbf{F}(\mathbf{m}_t)$, avoiding the expensive computation of the Jacobian of a neural network (Klushyn et al., 2021). We denote this locally-linear variant as NCDSSM-LL.

3.2. Inference

Exact inference in the model described above is intractable when the dynamics is non-linear and/or the observation emission distribution, $p(\mathbf{y}_k|\mathbf{a}_k)$, is modeled by arbitrary non-linear functions. In the modern deep learning context, a straightforward approach would be to approximate the posterior distribution over the states and auxiliary variables, $q(\mathbf{z}_{0:T}, \mathbf{a}_{0:T}|\mathbf{y}_{0:T})$, using recurrent neural networks (e.g., using ODE-RNNs when modeling in continuous time). However, such parameterizations have been shown to lead to poor optimization of the transition model in discrete-time SSMs, leading to inaccurate learning of system dynamics (Klushyn et al., 2021). Alternatively, directly applying continuous-discrete inference techniques to non-linear emission models requires computation of Jacobian matrices and inverses of $d \times d$ matrices (cf. Eq. 6) which scales poorly with the data dimensionality.

The introduction of linear-Gaussian auxiliary variables offers a middle ground between the two options above. It allows efficient use of continuous-discrete Bayesian inference techniques for the inference of states, avoiding fully amortized inference for auxiliary variables and states. Concretely, we split our inference procedure into two inference steps: (i) for auxiliary variables and (ii) for states.

Inference for auxiliary variables. We perform amortized inference for the auxiliary variables, factorizing the variational distribution as,

$$q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T}) = \prod_{k=0}^{T} q(\mathbf{a}_k|\mathbf{y}_k), \tag{14}$$

where $q(\mathbf{a}_k|\mathbf{y}_k) = \mathcal{N}(\mathbf{a}_k; f_\phi^\mu(\mathbf{y}_k), f_\phi^\Sigma(\mathbf{y}_k))$ and f_ϕ^μ, f_ϕ^Σ are neural networks. This can be viewed as the recognition network in a variational autoencoder, per timestep. This flexible factorization permits use of arbitrary recognition networks, thereby allowing arbitrary non-linear emission distributions, $p(\mathbf{y}_k|\mathbf{a}_k)$.

Inference for states. Given the variational distribution $q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T})$ in Eq. (14), we can draw samples, $\tilde{\mathbf{a}}_{0:T} \sim q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T})$, from it. Viewing $\tilde{\mathbf{a}}_{0:T}$ as pseudo-observations, we treat the remaining SSM (i.e., the states and auxiliary variables) separately. Specifically, conditioned on the auxiliary variables, $\tilde{\mathcal{A}}_\tau = \{\tilde{\mathbf{a}}_k : t_k \leq \tau\}$, we can answer inference queries over the states $\mathbf{z}_t$ in continuous time. This does not require additional inference networks and can be performed only using the generative model via classic continuous-discrete Bayesian inference techniques in Section 2. To infer the filtered density, $p_t(\mathbf{z}_t|\tilde{\mathcal{A}}_t)$, we can use Eq. (5) for the prediction step and Eq. (6) for the update step, replacing $\mathbf{y}_k$ by $\tilde{\mathbf{a}}_k$. Similarly, we can use Eq. (23) (Appendix) to infer the smoothed density, $p_t(\mathbf{z}_t|\tilde{\mathcal{A}}_T)$.

As the inference of states is now conditioned on auxiliary variables, only the inversion of $h \times h$ matrices is required which is computationally feasible as $\mathbf{a}_k$ generally has lower dimensionality than $\mathbf{y}_k$. Notably, this inference scheme does not require posterior SDEs for inference (as in other SDE-based models; cf. Section 4) and does not suffer from poor optimization of the transition model as we employ the (generative) transition model for the inference of states.

3.3. Learning

The parameters of the generative model $\{\theta\}$ and the inference network $\{\phi\}$ can be jointly optimized by maximizing the following evidence lower bound (ELBO) of the log-likelihood, $\log p_\theta(\mathbf{y}_{0:T})$,

$$\begin{aligned}&\log p_\theta(\mathbf{y}_{0:T})\\ &\geq \mathbb{E}_{q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T})}\left[\log \frac{\prod_{k=0}^{T} p_\theta(\mathbf{y}_k|\mathbf{a}_k)p_\theta(\mathbf{a}_{0:T})}{\prod_{k=0}^{T} q_\phi(\mathbf{a}_k|\mathbf{y}_k)}\right]\\ &=: \mathcal{L}_{\mathrm{ELBO}}(\theta,\phi). \end{aligned} \tag{15}$$

The distributions $p_\theta(\mathbf{y}_k|\mathbf{a}_k)$ and $q_\phi(\mathbf{a}_k|\mathbf{y}_k)$ in $\mathcal{L}_{\mathrm{ELBO}}$ are immediately available via the emission and recognition networks, respectively. What remains is the computation of $p_\theta(\mathbf{a}_{0:T})$. Fortunately, $p_\theta(\mathbf{a}_{0:T})$ can be computed as a byproduct of the inference (filtering) procedure described in Section 3.2. The distribution factorizes as

$$p(\mathbf{a}_{0:T}) = p(\mathbf{a}_0)\prod_{k=1}^{T} p(\mathbf{a}_k|\mathcal{A}_{t_{k-1}}),$$

where $p(\mathbf{a}_k|\mathcal{A}_{t_{k-1}}) = \mathcal{N}(\mathbf{a}_k; \mathbf{H}\mathbf{m}_k^-, \mathbf{S}_k)$, and $\mathbf{m}_k^-$ and $\mathbf{S}_k$ are computed during the prediction and update steps, respectively. The $p_\theta(\mathbf{a}_{0:T})$ term can be viewed as a "prior" over the auxiliary variables. However, unlike the fixed standard Gaussian prior in a vanilla variational autoencoder, $p_\theta(\mathbf{a}_{0:T})$ is a learned prior given by the marginalization of the states, $\mathbf{z}_t$, from the underlying SSM. Algorithm 1 summarizes the learning algorithm for a single time series; in practice, minibatches of time series are sampled from the dataset.

3.4. Stable Implementation

A naive implementation of the numerical integration of ODEs (Eqs. 5 and 23) and other operations (Eq. 6) results in unstable training and crashing due to violation of the positive definite constraint for the covariance matrices. Commonly employed tricks such as symmetrization, $\mathbf{P} = (\mathbf{P} + \mathbf{P}^\top)/2$, and addition of a small positive number (ϵ) to the diagonal elements, $\mathbf{P} = \mathbf{P} + \epsilon\mathbf{I}$, did not solve these training issues. Therefore, we implemented our algorithms in terms of square root (Cholesky) factors, which proved critical to the stable training of NCDSSM. Several square root factors' based inference algorithms have been previously proposed (Zonov, 2019; Jorgensen et al., 2007; Kailath et al., 2000, Ch. 12). In the following, we discuss our implementation which is based on Zonov (2019). Further discussion on implementation stability, particularly in the case of non-linear dynamics, can be found in Appendix B.3.

Algorithm 1 Learning in Neural Continuous-Discrete State Space Models

```
Require: Observations $\{(\mathbf{y}_k, t_k)\}_{k=0}^{T}$ and model parameters $\{\theta, \phi\}$.
repeat
    Compute $q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T})$ using Eq. (14).
    Sample $\tilde{\mathbf{a}}_{0:T} \sim q_\phi(\mathbf{a}_{0:T}|\mathbf{y}_{0:T})$.
    $\_\_, \log p_\theta(\tilde{\mathbf{a}}_{0:T}) \leftarrow \textsc{Filter}(\tilde{\mathbf{a}}_{0:T}, t_{0:T}; \theta)$
    ▷ cf. Algorithm 3 (Appendix) for $\textsc{Filter}$.
    Compute $\prod_{k=0}^{T} p_\theta(\mathbf{y}_k|\tilde{\mathbf{a}}_k)$ using Eq. (9).
    Optimize $\mathcal{L}_{\mathrm{ELBO}}(\theta, \phi)$.
until end of training.
```

We begin with a lemma that shows that the square root factor of the sum of two matrices with square root factors can be computed using QR decomposition.

Lemma 3.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two* $n \times n$ *matrices with square root factors* $\mathbf{A}^{1/2}$ *and* $\mathbf{B}^{1/2}$*, respectively. The matrix* $\mathbf{C} = \mathbf{A} + \mathbf{B}$ *also has a square root factor,* $\mathbf{C}^{1/2}$*, given by*

$$\Theta, \begin{bmatrix}\mathbf{C}^{1/2} & \mathbf{0}_{n\times n}\end{bmatrix}^\top = \mathrm{QR}\left(\begin{bmatrix}\mathbf{A}^{1/2} & \mathbf{B}^{1/2}\end{bmatrix}^\top\right),$$

where Θ *is the orthogonal* Q *matrix given by* QR *decomposition and* $\mathbf{0}_{n\times n}$ *is an* $n \times n$ *matrix of zeros.*

Prediction step. The solution of matrix differential equations of the form in Eq. (5b) — called Lyapunov differential equations — over $[t_0, t_1]$ is given by (Abou-Kandil et al., 2012, Corollary 1.1.6)

$$\mathbf{P}_{t_1} = \mathbf{\Phi}_{t_1}\mathbf{P}_{t_0}\mathbf{\Phi}_{t_1}^\top + \int_{t_0}^{t_1} \mathbf{\Phi}_t\mathbf{D}_t\mathbf{\Phi}_t^\top dt, \tag{16}$$

where $\mathbf{\Phi}_t$, called the fundamental matrix, is defined by

$$\frac{d\mathbf{\Phi}_t}{dt} = \mathbf{F}_{\mathbf{z}}(\mathbf{m}_t, t)\mathbf{\Phi}_t \;\text{ and }\; \mathbf{\Phi}_{t_0} = \mathbf{I}. \tag{17}$$

This initial value problem can be solved using an off-the-shelf ODE solver. Let $\{\tilde{\mathbf{\Phi}}_1 = \mathbf{I}, \tilde{\mathbf{\Phi}}_2, \dots, \tilde{\mathbf{\Phi}}_n\}$ be intermediate solutions of Eq. (17) given by an ODE solver with step size η, Eq. (16) can be approximated as

$$\begin{aligned}\mathbf{P}_{t_1} &\approx \tilde{\mathbf{\Phi}}_n\mathbf{P}_{t_0}\tilde{\mathbf{\Phi}}_n^\top\\ &+ \frac{\eta}{2}\left(\tilde{\mathbf{\Phi}}_1\mathbf{D}_1\tilde{\mathbf{\Phi}}_1^\top + 2\tilde{\mathbf{\Phi}}_2\mathbf{D}_2\tilde{\mathbf{\Phi}}_2^\top + \cdots + \tilde{\mathbf{\Phi}}_n\mathbf{D}_n\tilde{\mathbf{\Phi}}_n^\top\right).\end{aligned} \tag{18}$$

The additions in Eq. (18) are performed using Lemma 3.1 with square root factors $\tilde{\mathbf{\Phi}}_n\mathbf{P}_{t_0}^{1/2}$ and $\{\tilde{\mathbf{\Phi}}_j\mathbf{D}_j^{1/2}\}_{j=1}^{n}$.

Update step. Using similar arguments as in the proof of Lemma 3.1 (cf. Appendix B.3 for details), the update step (Eq. 6) can be performed by the QR decomposition of the square root factor

$$\begin{bmatrix} \mathbf{R}^{1/2} & \mathbf{H}(\mathbf{P}_k^-)^{1/2} \\ \mathbf{0}_{m\times d} & (\mathbf{P}_k^-)^{1/2} \end{bmatrix}^\top . \tag{19}$$

Let $\begin{bmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{Z} \end{bmatrix}^\top$ be the upper triangular R matrix obtained from the QR decomposition of (19). The square root factor of the updated covariance matrix, $\mathbf{P}_k^{1/2}$, and the Kalman gain matrix, $\mathbf{K}_k$, are then given by $\mathbf{P}_k^{1/2} = \mathbf{Z}$ and $\mathbf{K}_k = \mathbf{Y}\mathbf{X}^{-1}$, respectively.

4. Related Work

Several previous works (Chung et al., 2015; Krishnan et al., 2015; Karl et al., 2016; Krishnan et al., 2017; Doerr et al., 2018) have proposed SSM-like models for discrete-time sequential data, trained via amortized variational inference. Unlike NCDSSM, these models approximate sequential Bayesian inference (i.e., filtering and smoothing) via deterministic RNNs and are limited to the discrete time setting.

Bayesian inference for a subset of latent variables combined with amortized inference for others has previously been studied for SSMs. SNLDS (Dong et al., 2020) and REDSDS (Ansari et al., 2021) perform amortized inference for the states and exact inference for discrete random variables (switches and duration counts) in switching SSMs. KVAE (Fraccaro et al., 2017), EKVAE (Klushyn et al., 2021) and ARSGLS (Kurle et al., 2020) introduce auxiliary variables and perform classic Bayesian filtering and smoothing for the state variables, similar to NCDSSM. However, these models use specific parameterizations of state dynamics and operate on discrete-time sequential data. In contrast, we propose a general framework for continuous-time modeling of irregularly-sampled time series with multiple possible parameterizations of the dynamics.

Since the introduction of NeuralODE (Chen et al., 2018), various models based on neural differential equations have been proposed for continuous-time modeling of time series (Rubanova et al., 2019; De Brouwer et al., 2019; Yildiz et al., 2019; Li et al., 2020; Liu et al., 2020; Kidger et al., 2020; Solin et al., 2021). Amongst these, we focus on the latent variable models as they are closely related to SSMs.

LatentODE (Rubanova et al., 2019) encodes the entire context window into an initial state using an encoder (e.g., ODE-RNN) and uses a NeuralODE to model latent dynamics. ODE2VAE (Yildiz et al., 2019) decomposes the latent state into position and velocity components to explicitly model the acceleration and parameterize the ODE dynamics by Bayesian neural networks to account for uncertainty. LatentSDE (Li et al., 2020) uses a posterior SDE in the latent space to infer the latent dynamics together with a prior (generative) SDE in a variational setup. Solin et al. (2021) proposed a variant of LatentSDE trained by exploiting the Gaussian assumed density approximation of the non-linear SDE. VSDN (Liu et al., 2020) uses ODE-RNNs to provide historical information about the time series to the SDE drift and diffusion functions. These existing ODE-based models either use deterministic latent dynamics and/or create a restrictive bottleneck by encoding the entire time series into an initial state. The SDE-based models require posterior SDEs to infer the dynamics; new observations are incorporated in an ad-hoc fashion, potentially resulting in a disparity between posterior and generative dynamics and a non-Markovian state space. Contrary to previous models, NCDSSM uses (i) stochastic Markovian dynamics, (ii) incorporates observations via a principled Bayesian update, (iii) disentangles recognition from dynamics using auxiliary variables and (iv) performs continuous-discrete Bayesian inference for the state variables (dynamics), obviating the need for posterior SDEs.

5. Experiments

In this section, we present empirical results on time series imputation and forecasting tasks. Our primary focus was to investigate the models' ability to capture the underlying dynamics of the time series, gauged by the accuracy of long-term forecasts beyond the training context. We experimented with the three variants of our model described in Section 3.1: NCDSSM-LTI, NCDSSM-NL, and NCDSSM-LL. Our main baselines were LatentODE and LatentSDE, two popular continuous-time latent variable models with deterministic and stochastic dynamics, respectively. We also compared NCDSSM against several other baselines for individual experiments. We first discuss experiment results on the low-dimensional bouncing ball and damped pendulum datasets, then move to higher dimensional settings: walking sequences from the CMU Motion Capture (MoCap) dataset, the USHCN daily climate dataset, and two 32x32 dimensional video datasets (Box and Pong).

5.1. Bouncing Ball and Damped Pendulum

The bouncing ball and damped pendulum datasets have known ground truth dynamics, which facilitates quality assessment of the dynamics learned by a given model. For details on these datasets, please refer to Appendix C.1. In brief, the univariate bouncing ball dataset exhibits piecewise-linear dynamics, whilst bivariate damped pendulum dataset (Karl et al., 2016; Kurle et al., 2020) exhibits non-linear latent dynamics.

We trained all the models on 10s/5s sequences (with a discretization of 0.1s) for bouncing ball/damped pendulum with 0%, 30%, 50% and 80% timesteps missing at ran-

Table 1. Imputation and forecasting results for bouncing ball and damped pendulum datasets averaged over 50 sample trajectories. Mean ± standard deviation are computed over 5 independent runs.

Dataset	Model	Imputation MSE (↓) (% Missing)			Forecast MSE (↓) (% Missing)			
		30%	50%	80%	0%	30%	50%	80%
Bouncing Ball	LatentODE (Rubanova et al., 2019)	0.007 ± 0.000	0.008 ± 0.001	0.011 ± 0.000	0.386 ± 0.025	0.489 ± 0.133	0.422 ± 0.053	0.412 ± 0.048
	LatentSDE (Li et al., 2020)	**0.006 ± 0.000**	0.007 ± 0.000	0.011 ± 0.001	0.408 ± 0.043	1.209 ± 1.115	1.567 ± 2.263	0.352 ± 0.077
	NCDSSM-LTI	0.020 ± 0.001	0.026 ± 0.001	0.067 ± 0.002	0.592 ± 0.106	0.557 ± 0.014	0.556 ± 0.025	0.555 ± 0.022
	NCDSSM-NL	**0.006 ± 0.000**	**0.006 ± 0.000**	**0.007 ± 0.000**	**0.037 ± 0.018**	0.036 ± 0.007	**0.041 ± 0.007**	0.115 ± 0.029
	NCDSSM-LL	**0.006 ± 0.000**	**0.006 ± 0.000**	0.008 ± 0.001	**0.037 ± 0.028**	**0.034 ± 0.016**	0.049 ± 0.034	**0.076 ± 0.017**
Damped Pendulum	LatentODE (Rubanova et al., 2019)	0.151 ± 0.002	0.155 ± 0.002	0.206 ± 0.013	0.097 ± 0.042	0.117 ± 0.001	0.119 ± 0.001	0.148 ± 0.007
	LatentSDE (Li et al., 2020)	0.092 ± 0.076	0.148 ± 0.001	0.229 ± 0.001	0.046 ± 0.046	0.084 ± 0.058	0.147 ± 0.020	0.357 ± 0.096
	NCDSSM-LTI	0.036 ± 0.001	0.057 ± 0.001	0.120 ± 0.002	0.282 ± 0.084	1.017 ± 1.363	1.527 ± 1.440	0.231 ± 0.050
	NCDSSM-NL	**0.008 ± 0.000**	**0.011 ± 0.000**	**0.033 ± 0.002**	**0.011 ± 0.004**	0.011 ± 0.003	**0.012 ± 0.003**	**0.034 ± 0.019**
	NCDSSM-LL	**0.008 ± 0.000**	**0.011 ± 0.000**	0.037 ± 0.003	0.025 ± 0.030	**0.010 ± 0.001**	0.020 ± 0.008	0.055 ± 0.007

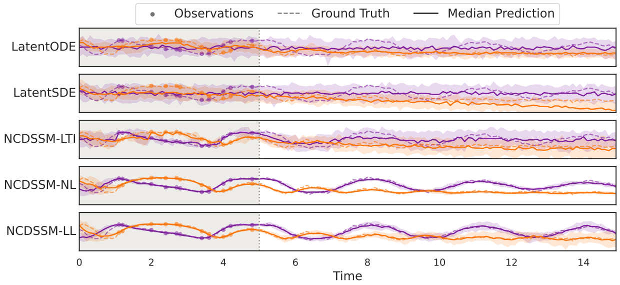


Figure 2. Predictions from different models on the damped pendulum dataset in the 80% missing data setting. The ground truth is shown using dashed lines with observed points in the context window (gray shaded region) shown as filled circles. The vertical dashed gray line marks the beginning of the forecast horizon. Solid lines indicate median predictions with 90% prediction intervals shaded around them. The purple and orange colors indicate observation dimensions. NCDSSM-NL and NCDSSM-LL are significantly better at forecasting compared to the baselines.

dom to simulate irregularly-sampled data. The models were evaluated on imputation of the missing timesteps and forecasts of 20s/10s beyond the training regime for bouncing ball/damped pendulum.

Table 1 reports the imputation and forecast mean squared error (MSE) for different missing data settings. In summary, the NCDSSM models with non-linear and locally-linear dynamics (NCDSSM-NL and NCDSSM-LL) perform well across datasets, settings, and random initializations, significantly outperforming the baselines. Furthermore, for these low-dimensional datasets, learning latent representations in the form of auxiliary variables is not required and we can set the recognition and emission functions in Eq. (14) and Eq. (9) to identity functions. This results in NCDSSM models requiring 2-5 times fewer parameters than LatentODE and LatentSDE (cf. Table 5 in the Appendix).

Fig. 2 shows example predictions from the best performing run of every model for 80% missing data for the pendulum (cf. Appendix D for other settings). NCDSSM-NL and NCDSSM-LL generates far better predictions both inside and outside the context window compared to the baselines. Ordinary least squares (OLS) goodness-of-fit results in Table 6 (Appendix) suggest that this performance can be attributed to our models having learnt the correct dynamics; latent states from NCDSSM-NL and NCDSSM-LL are highly correlated with the ground truth angle and angular velocity for all missingness scenarios. In other words, the models have learnt a Markovian state space which is informative about the dynamics at a specific time.

Table 2. Forecasting results for the CMU MoCap walking dataset averaged over 50 sample trajectories with 95% prediction interval based on the t-statistic in parentheses. †Baseline results from Solin et al. (2021).

Model	MSE (↓)	
	†Setup 1	Setup 2
npODE (Heinonen et al., 2018)	22.96	–
NeuralODE (Chen et al., 2018)	22.49 (0.88)	–
ODE^{2}VAE-KL (Yildiz et al., 2019)	8.09 (1.95)	–
LatentODE (Rubanova et al., 2019)	5.98 (0.28)	31.62 (0.05)
LatentSDE (Li et al., 2020)	**4.03 (0.20)**	9.52 (0.21)
LatentApproxSDE (Solin et al., 2021)	7.55 (0.05)	–
NCDSSM-LTI	13.90 (0.02)	5.22 (0.02)
NCDSSM-NL	5.69 (0.01)	6.73 (0.02)
NCDSSM-LL	9.96 (0.01)	**4.74 (0.01)**

5.2. CMU Motion Capture (Walking)

This dataset comprises walking sequences of subject 35 from the CMU MoCap database containing joint angles of subjects performing everyday activities. We used a preprocessed version of the dataset from Yildiz et al. (2019) that has 23 50-dimensional sequences of length 300. We tested the models under two setups. Setup 1 (Yildiz et al., 2019; Li et al., 2020; Solin et al., 2021) involves training on complete 300 timestep sequences from the training set and using only the first 3 timesteps as context to predict the remaining 297 timesteps during test time. Although challenging, this setup does not evaluate the model's performance beyond the training context. Thus, we propose Setup 2 in which we train the model only using the first 200 timesteps. During test time, we give the first 100 timesteps as context and predict the remaining 200 timesteps.

The forecast MSE results for both setups are reported in

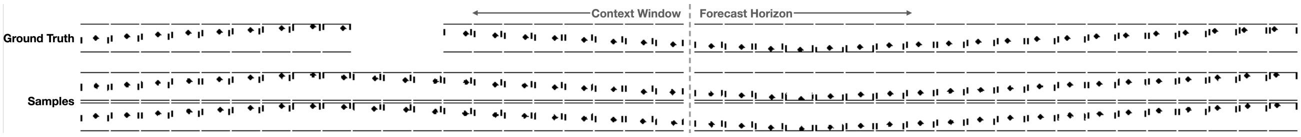


Figure 3. Sample predictions from NCDSSM-NL on the Pong dataset. The top row is the ground truth with some missing observations in the context window. The next two rows show trajectories sampled from NCDSSM-NL upto 20 forecast steps. NCDSSM-NL is able to both impute and forecast accurately. Best viewed zoomed-in on a computer. More examples in Appendix D.

Table 3. Forecasting results for the USHCN climate dataset. Mean ± standard deviation are computed over 5 folds as described in De Brouwer et al. (2019). †Results from De Brouwer et al. (2019). ‡Results from Liu et al. (2020).

Model	MSE (↓)
†NeuralODE-VAE (Chen et al., 2018)	0.83 ± 0.10
†SequentialVAE (Krishnan et al., 2015)	0.83 ± 0.07
†GRU-D (Che et al., 2018)	0.53 ± 0.06
†T-LSTM (Baytas et al., 2017)	0.59 ± 0.11
†GRUODE-B (De Brouwer et al., 2019)	0.43 ± 0.07
‡ODE-RNN (Rubanova et al., 2019)	0.39 ± 0.06
‡LatentODE (Rubanova et al., 2019)	0.77 ± 0.09
‡LatentSDE (Li et al., 2020)	0.74 ± 0.11
‡VSDN-F (IWAE) (Liu et al., 2020)	0.37 ± 0.06
NCDSSM-LTI	0.38 ± 0.07
NCDSSM-NL	**0.34 ± 0.06**
NCDSSM-LL	0.37 ± 0.06

Table 2. NCDSSM-NL performs better than all baselines except LatentSDE on Setup 1 while NCDSSM models perform significantly better than baselines on Setup 2. This showcases NCDSSM's ability to correctly model the latent dynamics, aiding accurate long-term predictions beyond the training context.

5.3. USHCN Climate Indicators

We evaluated the models on the United States Historical Climatology Network (USHCN) dataset that comprises measurements of five climate indicators across the United States. The preprocessed version of this dataset from De Brouwer et al. (2019) contains sporadic time series (i.e., with measurements missing *both* over the time and feature axes) from 1,114 meteorological stations over 4 years. Following De Brouwer et al. (2019), we trained the models on sequences from the training stations and evaluated them on the task of predicting the next 3 measurements given the first 3 years as context from the held-out test stations. The results in Table 3 show that NCDSSM-NL outperforms all the baselines with NCDSSM-LTI and NCDSSM-LL performing better than most of the baselines.

5.4. Pymunk Physical Environments

Finally, we evaluated the models on two high-dimensional (video) datasets of physical environments used in Fraccaro et al. (2017), simulated using the Pymunk Physics engine (Blomqvist, 2022): Box and Pong. The box dataset consists of videos of a ball moving in a 2-dimensional box and the pong dataset consists of videos of a Pong-like environment where two paddles move to keep a ball in the frame at all times. Each frame is a 32x32 binary image.

Table 4. Forecasting results for the Box and Pong datasets averaged over 16 sample trajectories.

Model	EMD (↓)	
	Box	Pong
LatentODE (Rubanova et al., 2019)	1.792	4.543
LatentSDE (Li et al., 2020)	1.925	3.505
NCDSSM-LTI	1.685	3.265
NCDSSM-NL	0.692	**1.714**
NCDSSM-LL	**0.632**	1.891

We trained the models on sequences of 20 frames with 20% of these frames randomly dropped. At test time, the models were evaluated on forecasts of 40 frames beyond the training context. For evaluation, we treat each image as a probability distribution on the XY-plane and report the earth mover's distance (EMD) between the ground truth and predicted images, averaged over the forecast horizon, in Table 4. NCDSSM-NL and NCDSSM-LL significantly outperform baseline models on both box and pong datasets. Fig. 6 (Appendix) shows the variation of EMD against time for different models. In the context window (0-2s), all models have EMD close to 0; however, in the forecast horizon (2-6s), the EMD rises rapidly and irregularly for LatentODE and LatentSDE but does so gradually for NCDSSM-NL and NCDSSM-LL. This indicates that the dynamics models learned by NCDSSM-NL and NCDSSM-LL are both accurate and robust.

Qualitatively, both NCDSSM-LL and NCDSSM-NL correctly impute the missing frames and the forecasts generated by them are similar to ground truth. Fig. 3 shows sample predictions for the pong dataset generated by NCDSSM-NL. In contrast, other models only impute the missing frames correctly, failing to generate accurate forecasts (cf. Appendix D).

6. Conclusion

In this work, we proposed a model for continuous-time modeling of irregularly-sampled time series. NCDSSM improves continuous-discrete SSMs with neural network-based parameterizations of dynamics, and modern inference and learning techniques. Through the introduction of auxiliary variables, NCDSSM enables efficient modeling of high-dimensional time series while allowing accurate continuous-discrete Bayesian inference of the dynamic states. Experiments on a variety of low- and high-dimensional datasets show that NCDSSM outperforms existing models on time series imputation and forecasting tasks.

Acknowledgements

This research is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-016). We thank Richard Kurle, Fabian Falck, Alexej Klushyn, and Marcel Kollovieh for helpful discussions and feedback.

References

Abou-Kandil, H., Freiling, G., Ionescu, V., and Jank, G. *Matrix Riccati equations in control and systems theory*. Birkhäuser, 2012.

Anderson, B. D. Fixed interval smoothing for nonlinear continuous time systems. *Information and Control*, 20 (3):294–300, 1972.

Ansari, A. F., Benidis, K., Kurle, R., Turkmen, A. C., Soh, H., Smola, A. J., Wang, B., and Januschowski, T. Deep explicit duration switching models for time series. *Advances in Neural Information Processing Systems*, 34, 2021.

Baytas, I. M., Xiao, C., Zhang, X., Wang, F., Jain, A. K., and Zhou, J. Patient subtyping via time-aware LSTM networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 65–74, 2017.

Blomqvist, V. Pymunk, 11 2022. URL https://pymunk.org.

Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A recurrent latent variable model for sequential data. *Advances in neural information processing systems*, 28, 2015.

De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. *Advances in neural information processing systems*, 32, 2019.

Doerr, A., Daniel, C., Schiegg, M., Duy, N.-T., Schaal, S., Toussaint, M., and Sebastian, T. Probabilistic recurrent state-space models. In *International Conference on Machine Learning*, pp. 1280–1289. PMLR, 2018.

Dong, Z., Seybold, B., Murphy, K., and Bui, H. Collapsed amortized variational inference for switching nonlinear dynamical systems. In *International Conference on Machine Learning*, pp. 2638–2647. PMLR, 2020.

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., Tavenard, R., Tong, A., and Vayer, T. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021. URL http://jmlr.org/papers/v22/20-451.html.

Fraccaro, M., Kamronn, S., Paquet, U., and Winther, O. A disentangled recognition and nonlinear dynamics model for unsupervised learning. *arXiv preprint arXiv:1710.05741*, 2017.

Heinonen, M., Yildiz, C., Mannerström, H., Intosalmi, J., and Lähdesmäki, H. Learning unknown ODE models with Gaussian processes. In *International Conference on Machine Learning*, pp. 1959–1968. PMLR, 2018.

Jazwinski, A. H. *Stochastic processes and filtering theory*. Academic Press, 1970.

Jorgensen, J. B., Thomsen, P. G., Madsen, H., and Kristensen, M. R. A computationally efficient and robust implementation of the continuous-discrete extended Kalman filter. In *2007 American Control Conference*, pp. 3706–3712. IEEE, 2007.

Kailath, T., Sayed, A. H., and Hassibi, B. *Linear estimation*. Prentice Hall, 2000.

Karl, M., Soelch, M., Bayer, J., and Van der Smagt, P. Deep variational Bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Kidger, P., Morrill, J., Foster, J., and Lyons, T. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33: 6696–6707, 2020.

Klushyn, A., Kurle, R., Soelch, M., Cseke, B., and van der Smagt, P. Latent matters: Learning deep state-space models. *Advances in Neural Information Processing Systems*, 34:10234–10245, 2021.

Krishnan, R., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Krishnan, R. G., Shalit, U., and Sontag, D. Deep Kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Kurle, R., Rangapuram, S. S., de Bézenac, E., Günnemann, S., and Gasthaus, J. Deep rao-blackwellised particle filters for time series forecasting. *Advances in Neural Information Processing Systems*, 33, 2020.

Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882. PMLR, 2020.

Liu, Y., Xing, Y., Yang, X., Wang, X., Shi, J., Jin, D., and Chen, Z. Learning continuous-time dynamics by stochastic differential networks. *arXiv preprint arXiv:2006.06145*, 2020.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Øksendal, B. Stochastic differential equations. In *Stochastic differential equations*, pp. 65–84. Springer, 2003.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., and Januschowski, T. Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31, 2018.

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Särkkä, S. and Sarmavuori, J. Gaussian filtering and smoothing for continuous-discrete dynamic systems. *Signal Processing*, 93(2):500–510, 2013.

Särkkä, S. and Solin, A. *Applied stochastic differential equations*. Cambridge University Press, 2019.

Solin, A., Tamir, E., and Verma, P. Scalable inference in SDEs by direct matching of the Fokker–Planck–Kolmogorov equation. *Advances in Neural Information Processing Systems*, 34:417–429, 2021.

Yildiz, C., Heinonen, M., and Lahdesmaki, H. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Zonov, S. Kalman filter based sensor placement for Burgers equation. Master's thesis, University of Waterloo, 2019.

A. Proofs

A.1. Proof of Lemma 3.1

Lemma A.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two* $n \times n$ *matrices with square root factors* $\mathbf{A}^{1/2}$ *and* $\mathbf{B}^{1/2}$*, respectively. The matrix* $\mathbf{C} = \mathbf{A} + \mathbf{B}$ *also has a square root factor,* $\mathbf{C}^{1/2}$*, given by*

$$\Theta, \begin{bmatrix}\mathbf{C}^{1/2} & \mathbf{0}_{n\times n}\end{bmatrix}^\top = \mathrm{QR}\left(\begin{bmatrix}\mathbf{A}^{1/2} & \mathbf{B}^{1/2}\end{bmatrix}^\top\right),$$

where Θ *is the orthogonal* Q *matrix given by* QR *decomposition and* $\mathbf{0}_{n\times n}$ *is an* $n \times n$ *matrix of zeros.*

Proof. Our proof is based on Zonov (2019, Thm. 3.2). Consider the square root factor

$$\mathbf{Y} = \begin{bmatrix}\mathbf{A}^{1/2} & \mathbf{B}^{1/2}\end{bmatrix}.$$

Clearly, $\mathbf{C} = \mathbf{Y}\mathbf{Y}^\top$; however, we also have $\mathbf{C} = \mathbf{Y}\Theta\Theta^\top\mathbf{Y}^\top$, for any orthogonal matrix Θ. Thus, $\mathbf{Y}\Theta$ is also a square root factor of $\mathbf{C}$. Let Θ be an orthogonal matrix such that

$$\mathbf{Y}\Theta = \begin{bmatrix}\mathbf{X} & \mathbf{0}_{n\times n}\end{bmatrix}, \tag{20}$$

where $\mathbf{X}$ is an $n \times n$ lower triangular matrix. This implies that $\mathbf{X}$ is a square root factor of $\mathbf{C}$.

From Eq. (20), we further have the following,

$$\mathbf{Y} = \begin{bmatrix}\mathbf{X} & \mathbf{0}_{n\times n}\end{bmatrix}\Theta^\top, \tag{21}$$

$$\mathbf{Y}^\top = \Theta\begin{bmatrix}\mathbf{X}^\top \\ \mathbf{0}_{n\times n}\end{bmatrix}, \tag{22}$$

where we post-multiply by $\Theta^\top$ in the first step and use the fact that $\Theta\Theta^\top = \mathbf{I}$, and transpose both sides in the second step. We have thus expressed $\mathbf{Y}^\top$ as the product of an orthogonal matrix, Θ, and an upper triangular matrix, $\begin{bmatrix}\mathbf{X} & \mathbf{0}_{n\times n}\end{bmatrix}^\top$. Such a factorization can be performed by QR decomposition. Thus, we can compute the square root factor $\mathbf{C}^{1/2} = \mathbf{X}$ via the QR decomposition of $\begin{bmatrix}\mathbf{A}^{1/2} & \mathbf{B}^{1/2}\end{bmatrix}^\top$. □

Algorithm 2 Sum of Square Root Factors

```
function SumMatrixSqrts($\mathbf{A}^{1/2}$, $\mathbf{B}^{1/2}$)
    $\_, \begin{bmatrix}\mathbf{C}^{1/2} & \mathbf{0}_{n\times n}\end{bmatrix}^\top = \mathrm{QR}\left(\begin{bmatrix}\mathbf{A}^{1/2} & \mathbf{B}^{1/2}\end{bmatrix}^\top\right)$
    return $\mathbf{C}^{1/2}$
end function
```

B. Technical Details

B.1. Continuous-Discrete Bayesian Smoothing

Several approximate smoothing procedures based on Gaussian assumed density approximation have been proposed in the literature. We refer the reader to Särkkä & Sarmavuori (2013) for an excellent review of continuous-discrete smoothers. In the following, we discuss the *Type II extended RTS smoother* which is linear in the smoothing solution. According to this smoother, the mean, $\mathbf{m}_t^s$, and covariance matrix, $\mathbf{P}_t^s$, of the Gaussian approximation to the smoothing density, $p_t(\mathbf{z}_t|\mathcal{Y}_T)$, follow the backward ODEs,

$$\frac{d\mathbf{m}_t^s}{dt} = \mathbf{f}(\mathbf{m}_t, t) + \mathbf{C}(\mathbf{m}_t, t)(\mathbf{m}_t^s - \mathbf{m}_t), \tag{23a}$$

$$\frac{d\mathbf{P}_t^s}{dt} = \mathbf{C}(\mathbf{m}_t, t)\mathbf{P}_t^s + \mathbf{P}_t^s\mathbf{C}^\top(\mathbf{m}_t, t) - \mathbf{D}(\mathbf{m}_t, t), \tag{23b}$$

where $(\mathbf{m}_t, \mathbf{P}_t)$ is the filtering solution given by Eq. (5), $\mathbf{C}(\mathbf{m}_t, t) = \mathbf{F}_\mathbf{z}(\mathbf{m}_t, t) + \mathbf{D}(\mathbf{m}_t, t)\mathbf{P}_t^{-1}$ and backward means that the ODEs are solved backwards in time from the filtering solution ($\mathbf{m}_T^s = \mathbf{m}_T$, $\mathbf{P}_T^s = \mathbf{P}_T$).

B.2. Algorithms

In this section, we discuss the *stable* filtering and smoothing algorithms used in NCDSSM.

Algorithm 2 provides a utility function — SUMMATRIXSQRTS — that uses Lemma 3.1 to compute the square root factor of the sum of two matrices with square root factors. The square root factor version of the continuous-discrete Bayesian filtering algorithm is given in Algorithm 3. Note that the PREDICT step for linear time-invariant dynamics can be performed analytically using matrix exponentials (Särkkä & Solin, 2019, Ch. 6). We used the analytic solver for some of our experiments. The Type II RTS smoothing algorithm (Algorithm 4) takes the filtered distributions as input and computes the smoothed distribution at every filtered timestep. To compute the smoothed distribution between observed timesteps,

Algorithm 3 Continuous-Discrete Bayesian Filtering

```
function UPDATE($\mathbf{a}_k, \mathbf{m}_k^-, (\mathbf{P}_k^-)^{1/2}; \mathbf{H}, \mathbf{R}$)
    $\mathbf{R}^{1/2} \leftarrow \text{cholesky}(\mathbf{R})$
    $\mathbf{A} \leftarrow \begin{bmatrix} \mathbf{R}^{1/2} & \mathbf{H}(\mathbf{P}_k^-)^{1/2} \\ \mathbf{0}_{m\times d} & (\mathbf{P}_k^-)^{1/2} \end{bmatrix}$
    $\_, \begin{bmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{Z} \end{bmatrix}^\top \leftarrow \text{QR}(\mathbf{A}^\top)$
    $\mathbf{K}_k \leftarrow \mathbf{Y}\mathbf{X}^{-1}$
    $\hat{\mathbf{a}}_k \leftarrow \mathbf{H}\mathbf{m}_k^-$
    $\mathbf{m}_k \leftarrow \mathbf{m}_k^- + \mathbf{K}_k(\mathbf{a}_k - \hat{\mathbf{a}}_k)$
    $\mathbf{P}_k^{1/2} \leftarrow \mathbf{Z}$
    $\mathbf{S}_k^{1/2} \leftarrow \mathbf{X}$
    return $\mathbf{m}_k, \mathbf{P}_k^{1/2}, \hat{\mathbf{a}}_k, \mathbf{S}_k^{1/2}$
end function

function PREDICT($\mathbf{m}_k, \mathbf{P}_k^{1/2}, t_k, t_{k+1}; \mathbf{f}, \mathbf{Q}, \mathbf{G}$)
    $\mathbf{\Phi}_1 \leftarrow \mathbf{I}$
    $\{\tilde{\mathbf{m}}_j\}_{j=1}^n \leftarrow \text{odeint}\left(\frac{d\mathbf{m}_t}{dt} = \mathbf{f}(\mathbf{m}_t, t), \mathbf{m}_k, [\tau_1 = t_k, \ldots, \tau_n = t_{k+1}]\right)$
    $\{\tilde{\mathbf{\Phi}}_j\}_{j=1}^n \leftarrow \text{odeint}\left(\frac{d\mathbf{\Phi}_t}{dt} = \mathbf{F}_\mathbf{z}(\mathbf{m}_t, t)\mathbf{\Phi}_t, \mathbf{\Phi}_1, [\tau_1 = t_k, \ldots, \tau_n = t_{k+1}]\right)$
    ▷ the two coupled ODEs above are solved together.
    $\mathbf{m}_{k+1}^- \leftarrow \tilde{\mathbf{m}}_n$
    $(\mathbf{P}_{k+1}^-)^{1/2} \leftarrow$ REDUCESUMMATRIXSQRTS($\left[\tilde{\mathbf{\Phi}}_n\mathbf{P}_k^{1/2}, \sqrt{\frac{\eta}{2}}\tilde{\mathbf{\Phi}}_1\mathbf{D}_{\tau_1}^{1/2}, \sqrt{\eta}\tilde{\mathbf{\Phi}}_2\mathbf{D}_{\tau_2^{1/2}}, \ldots, \sqrt{\frac{\eta}{2}}\tilde{\mathbf{\Phi}}_n\mathbf{D}_{\tau_n}^{1/2}\right]$)
    ▷ REDUCESUMMATRIXSQRTS uses the SUMMATRIXSQRTS function in Algorithm 2, reducing it over the list.
    return $\mathbf{m}_{k+1}^-, (\mathbf{P}_{k+1}^-)^{1/2}$
end function

function FILTER($\mathbf{a}_{0:T}, t_{0:T}; \theta$)
    $\boldsymbol{\mu}_0, \mathbf{\Sigma}_0, \mathbf{f}, \mathbf{Q}, \mathbf{G}, \mathbf{H}, \mathbf{R} \leftarrow \theta$
    $\mathbf{m}_0^-, (\mathbf{P}_0^-)^{1/2} \leftarrow \boldsymbol{\mu}_0, \text{cholesky}(\mathbf{\Sigma}_0)$
    $\ell \leftarrow 0$
    for $i \leftarrow 0, T$ do
        $\mathbf{m}_i, \mathbf{P}_i^{1/2}, \hat{\mathbf{a}}_i, \mathbf{S}_i^{1/2} \leftarrow$ UPDATE($\mathbf{a}_i, \mathbf{m}_i^-, (\mathbf{P}_i^-)^{1/2}; \mathbf{H}, \mathbf{R}$)
        $\ell \leftarrow \ell + \log \mathcal{N}(\mathbf{a}_i; \hat{\mathbf{a}}_i, \mathbf{S}_i)$
        if $i = T$ then
            break
        end if
        $\mathbf{m}_{i+1}^-, (\mathbf{P}_{i+1}^-)^{1/2} \leftarrow$ PREDICT($\mathbf{m}_i, \mathbf{P}_i^{1/2}, t_i, t_{i+1}; \mathbf{f}, \mathbf{Q}, \mathbf{G}$)
    end for
    return $\{\mathbf{m}_i, \mathbf{P}_i^{1/2}\}_{i=0}^T, \ell$
end function
```

we cache the filtered distributions at these timesteps and provide them to the SMOOTH function together with the filtered distributions at observed timesteps.

Algorithm 4 Continuous-Discrete Type II Extended RTS Smoothing

```
function SmoothStep($\mathbf{m}_k^s$, $(\mathbf{P}_k^s)^{1/2}$, $\mathbf{m}_k$, $\mathbf{P}_k^{1/2}$, $t_k$, $t_{k-1}$; $\mathbf{f}, \mathbf{Q}, \mathbf{G}$)
    $\mathbf{\Phi}_1^s \leftarrow \mathbf{I}$
    $\{\tilde{\mathbf{m}}_j^s\}_{j=1}^n \leftarrow \text{odeint}\left(\frac{d\mathbf{m}_t^s}{dt} = \mathbf{f}(\mathbf{m}_k, t) + \mathbf{C}(\mathbf{m}_k, t)(\mathbf{m}_t^s - \mathbf{m}_k), \mathbf{m}_k^s, [\tau_1 = t_k, \dots, \tau_n = t_{k-1}]\right)$
    $\{\tilde{\mathbf{\Phi}}_j^s\}_{j=1}^n \leftarrow \text{odeint}\left(\frac{d\mathbf{\Phi}_t^s}{dt} = \mathbf{C}(\mathbf{m}_k, t)\mathbf{\Phi}_t^s, \mathbf{\Phi}_1^s, [\tau_1 = t_k, \dots, \tau_n = t_{k-1}]\right)$
    ▷ the two coupled ODEs above are solved together.
    $\mathbf{m}_{k-1}^s \leftarrow \tilde{\mathbf{m}}_n^s$
    $(\mathbf{P}_{k-1}^s)^{1/2} \leftarrow$ ReduceSumMatrixSqrts($\left[\tilde{\mathbf{\Phi}}_n^s \mathbf{P}_k^{1/2}, \sqrt{\frac{\eta}{2}}\tilde{\mathbf{\Phi}}_1^s \mathbf{D}_{\tau_1}^{1/2}, \sqrt{\eta}\tilde{\mathbf{\Phi}}_2^s \mathbf{D}_{\tau_2^{1/2}}, \dots, \sqrt{\frac{\eta}{2}}\tilde{\mathbf{\Phi}}_n^s \mathbf{D}_{\tau_n}^{1/2}\right]$)
    ▷ ReduceSumMatrixSqrts uses the SumMatrixSqrts function in Algorithm 2, reducing it over the list.
    return $\mathbf{m}_{k-1}^s$, $(\mathbf{P}_{k-1}^s)^{1/2}$
end function

function Smooth($\{\mathbf{m}_i, \mathbf{P}_i^{1/2}\}_{i=0}^T$, $t_{0:T}$; $\theta$)
    $\boldsymbol{\mu}_0, \mathbf{\Sigma}_0, \mathbf{f}, \mathbf{Q}, \mathbf{G}, \mathbf{H}, \mathbf{R} \leftarrow \theta$
    $\mathbf{m}_T^s, (\mathbf{P}_T^s)^{1/2} \leftarrow \mathbf{m}_T, \mathbf{P}_T^{1/2}$
    for $i \leftarrow T, 1$ do                                   ▷ note the time reversal.
        $\mathbf{m}_{i-1}^s, (\mathbf{P}_{i-1}^s)^{1/2} \leftarrow$ SmoothStep($\mathbf{m}_i^s$, $(\mathbf{P}_i^s)^{1/2}$, $\mathbf{m}_{i-1}$, $\mathbf{P}_{i-1}^{1/2}$, $t_i$, $t_{i-1}$; $\mathbf{f}, \mathbf{Q}, \mathbf{G}$)
    end for
    return $\{\mathbf{m}_i^s, (\mathbf{P}_i^s)^{1/2}\}_{i=0}^T$
end function
```

B.3. Stable Implementation (Contd.)

Square Root Factor Measurement Update. In Section 3.4, we discussed a square root factor version of the measurement update step via the QR decomposition of $\mathbf{A}^\top$, where,

$$\mathbf{A} = \begin{bmatrix} \mathbf{R}^{1/2} & \mathbf{H}(\mathbf{P}_k^-)^{1/2} \\ \mathbf{0}_{m \times d} & (\mathbf{P}_k^-)^{1/2} \end{bmatrix}. \tag{24}$$

Let $\Theta, \mathbf{U} = \mathrm{QR}(\mathbf{A}^\top)$, where

$$\mathbf{U} = \begin{bmatrix} \mathbf{X} & \mathbf{0} \\ \mathbf{Y} & \mathbf{Z} \end{bmatrix}^\top. \tag{25}$$

In the following, we show how $\mathbf{P}_k^{1/2} = \mathbf{Z}$. Our proof is based on Zonov (2019) and we refer the reader to Zonov (2019, Appendix A) for the proof of $\mathbf{K}_k = \mathbf{Y}\mathbf{X}^{-1}$.

Proof. Note that $\mathbf{A}$ is a square root factor of

$$\begin{bmatrix} \mathbf{R} + \mathbf{H}\mathbf{P}_k^-\mathbf{H}^\top & \mathbf{H}\mathbf{P}_k^- \\ (\mathbf{P}_k^-)^\top\mathbf{H}^\top & \mathbf{P}_k^- \end{bmatrix}. \tag{26}$$

Matching the terms in (26) with the terms in

$$\mathbf{U}\mathbf{U}^\top = \begin{bmatrix} \mathbf{X}\mathbf{X}^\top & \mathbf{X}\mathbf{Y}^\top \\ \mathbf{Y}\mathbf{X}^\top & \mathbf{Y}\mathbf{Y}^\top + \mathbf{Z}\mathbf{Z}^\top \end{bmatrix}, \tag{27}$$

we get the following equations,

$$\mathbf{X}\mathbf{X}^\top = \mathbf{R} + \mathbf{H}\mathbf{P}_k^- \mathbf{H}^\top, \tag{28a}$$
$$\mathbf{X}\mathbf{Y}^\top = \mathbf{H}\mathbf{P}_k^-, \tag{28b}$$
$$\mathbf{Y}\mathbf{X}^\top = (\mathbf{P}_k^-)^\top \mathbf{H}^\top, \tag{28c}$$
$$\mathbf{Y}\mathbf{Y}^\top + \mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^-. \tag{28d}$$

From Eq. (28d), we have the following,

$$\mathbf{Y}\mathbf{Y}^\top + \mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^-, \tag{29a}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{Y}\mathbf{Y}^\top, \tag{29b}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{Y}(\mathbf{X}^\top\mathbf{X}^{-\top})(\mathbf{X}^{-1}\mathbf{X})\mathbf{Y}^\top, \tag{29c}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y}^\top, \tag{29d}$$

where we introduce $\mathbf{I} = (\mathbf{X}^\top\mathbf{X}^{-\top})(\mathbf{X}^{-1}\mathbf{X})$ in the third step and use the property $(\mathbf{X}\mathbf{X}^\top)^{-1} = \mathbf{X}^{-\top}\mathbf{X}^{-1}$ in the last step. Substituting values from Eq. (28), we get,

$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - (\mathbf{P}_k^-)^\top\mathbf{H}^\top\mathbf{S}_k^{-1}\mathbf{H}\mathbf{P}_k^- \tag{30a}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - (\mathbf{P}_k^-)^\top\mathbf{H}^\top\mathbf{S}_k^{-1}(\mathbf{S}_k\mathbf{S}_k^{-1})\mathbf{H}\mathbf{P}_k^- \tag{30b}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{P}_k^-\mathbf{H}^\top\mathbf{S}_k^{-1}\mathbf{S}_k\mathbf{S}_k^{-\top}\mathbf{H}(\mathbf{P}_k^-)^\top \tag{30c}$$
$$\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top \tag{30d}$$

where we introduce $\mathbf{I} = \mathbf{S}_k\mathbf{S}_k^{-1}$ in the second step, use the fact that $\mathbf{S}_k$ is symmetric in the third step and substitute the value of $\mathbf{K}_k$ from Eq. (6b) in last step. Note that $\mathbf{Z}\mathbf{Z}^\top = \mathbf{P}_k^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top = \mathbf{P}_k$; therefore, $\mathbf{Z} = \mathbf{P}_k^{1/2}$. □

Regularizing Non-Linear Dynamics. We now discuss the techniques we employed to regularize the latent dynamics in NCDSSM. Particularly in the case of non-linear dynamics (NCDSSM-NL), regularization is critical for stable training. The drift function, $\mathbf{f}$, was parameterized by an MLP in all our experiments. We experimented with the $\tanh$ and $\mathrm{softplus}$ non-linearities. We found that applying the non-linearity after the last layer was important when using $\tanh$. Furthermore, we also initialized the parameters of the last layer to 0 when using $\tanh$. In the case of experiments with a large time interval (e.g., MoCap and USHCN), application of spectral normalization (Miyato et al., 2018) along with the $\mathrm{softplus}$ non-linearity proved critical for stable training. In the following, we present our hypothesis on why spectral normalization stabilizes training.

According to Øksendal (2003, Section 5.2), one of the conditions for the existence of a unique solution of an SDE is the Lipschitz continuity of the drift function, $\mathbf{f}$. Applying spectral normalization regularizes the neural network to be 1-Lipschitz, aiding its solvability using numerical methods. However, spectral normalization is even more important in the case of NCDSSM from a practical perspective — it prevents the numerical explosion of the elements of $\mathbf{\Phi}_t$ in the prediction step (Eq. 17), as discussed below.

Consider the case of a fixed Jacobian matrix $\mathbf{F}_\mathbf{z}$ in an interval $[t_1, t_2]$. In this case, the solution of Eq. (17) is given by

$$\mathbf{\Phi}_{t_2} = \exp\left(\mathbf{F}_\mathbf{z}(t_2 - t_1)\right)\mathbf{\Phi}_{t_1}, \tag{31}$$

where $\exp\left(\mathbf{F}_\mathbf{z}(t_2 - t_1)\right)$ denotes the matrix exponential. For unregularized drifts, the elements of $\exp\left(\mathbf{F}_\mathbf{z}(t_2 - t_1)\right)$ can become arbitrarily large. However, in the case of 1-Lipschitz drift functions (as provided by spectral normalization), the spectral norm of $\exp\left(\mathbf{F}_\mathbf{z}\right)$ is bounded by $\exp(1)$, as shown in Lemma B.1. This controls the growth rate of the elements of fundamental matrix, $\mathbf{\Phi}_t$.

Lemma B.1. *Let* $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^m$ *be a 1-Lipschitz function and* $\mathbf{J}_\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^{m\times m}$ *be its Jacobian function. Then,* $\|\exp(\mathbf{J}_\mathbf{g}(\mathbf{z}))\|_2 \leq \exp(1)\ \forall\, \mathbf{z} \in \mathbb{R}^m$ *where* $\|\cdot\|_2$ *denotes the spectral norm of a matrix.*

Proof. The spectral norm of the Jacobian of a K-Lipschitz function is bounded by K. Thus, we have,

$$\|\mathbf{J}_\mathbf{g}(\mathbf{z})\|_2 \leq 1\ \forall\, \mathbf{z} \in \mathbb{R}^m. \tag{32}$$

Using the power series representation of the matrix exponential,

$$\exp(\mathbf{A}) = \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!},$$

we get the following bound on $\|\exp(\mathbf{J_g}(\mathbf{z}))\|_2$,

$$\|\exp(\mathbf{J_g}(\mathbf{z}))\|_2 \leq \sum_{k=0}^{\infty} \left\| \frac{\mathbf{J_g}(\mathbf{z})^k}{k!} \right\|_2 \leq \sum_{k=0}^{\infty} \frac{\|\mathbf{J_g}(\mathbf{z})\|_2^k}{k!} = \exp(\|\mathbf{J_g}(\mathbf{z})\|_2). \tag{33}$$

Combining Eq. (33) with Eq. (32), we get,

$$\|\exp(\mathbf{J_g}(\mathbf{z}))\|_2 \leq \exp(\|\mathbf{J_g}(\mathbf{z})\|_2) \leq \exp(1), \tag{34}$$

which completes the proof. □

For the same reasons as discussed above, we initialized the transition matrices in our linear models to be random orthogonal matrices as orthogonal matrices have spectral norm equal to 1. However, in the case of NCDSSM-LTI on the USHCN dataset, this initialization was not sufficient during the initial phase of training and we used a random skew-symmetric matrix instead. The matrix exponential of a skew-symmetric matrix is an orthogonal matrix. We generated a random skew-symmetric matrix as follows,

$$\begin{aligned} \mathbf{F} &\sim [\mathcal{N}(0,1)]^{m\times m}, \\ \mathbf{F} &= \left(\frac{\mathbf{F} - \mathbf{F}^\top}{2}\right). \end{aligned}$$

Fixed Measurement Matrix. We used a fixed rectangular identity matrix as the auxiliary measurement matrix ($\mathbf{H}$ in Eq. 8) in our bouncing ball, damped pendulum and CMU MoCap (walking) experiments as it lead to improved learning of dynamics. This parameterization forces the model to learn the static (e.g., position) and dynamic (e.g., velocity) components in separate elements of the latent state, thereby disentangling them (Klushyn et al., 2021).

B.4. Imputation and Forecasting

In this section, we describe how to perform imputation and forecasting using a trained NCDSSM.

For imputation, the timesteps at which imputation is to be performed are provided to the FILTER function during filtering. The filtered distributions are then passed to the SMOOTH function and (imputed) samples are drawn from the smoothed distributions.

For forecasting, filtering is first performed over the context time series. The PREDICT function is then used up to end of the forecast horizon, starting from the last filtered distribution. Sample forecast trajectories are then drawn from these predicted distributions.

C. Experiment Details

C.1. Datasets

Bouncing Ball and Damped Pendulum. The bouncing ball dataset comprises univariate time series of the position of a ball bouncing between two fixed walls, in the absence of dissipative forces. The initial position, x_0, and velocity, v_0, of the ball are chosen at random, as follows,

$$x_0 \sim \mathcal{U}(-1, 1), \tag{35}$$

$$v_0 \sim \mathcal{U}(0.05, 0.5) \times \mathcal{U}\{-1, 1\}, \tag{36}$$

where $\mathcal{U}(a,b)$ denotes a uniform distribution on (a,b) and $\mathcal{U}\{c_1,\dots,c_k\}$ denotes a uniform categorical distribution on $\{c_1,\dots,c_k\}$. The observed position, y_k, is a corrupted version of the true position, x_k,

$$y_k \sim \mathcal{N}(x_k, 0.05^2). \tag{37}$$

Collisions with the walls, located at -1 and $+1$, are assumed to be perfectly elastic, i.e., the sign of the velocity gets flipped when the ball hits either of the walls. Thus, the ball exhibits piecewise-linear dynamics. We used the Euler integrator with a step size of 0.1s to simulate the dynamics. The training, validation, and test datasets consist of 5000, 500, and 500 sequences of length 30s each, respectively.

The damped pendulum dataset (Karl et al., 2016; Kurle et al., 2020) comprises bivariate time series of the XY-coordinates of a pendulum oscillating in the presence of a damping force. The non-linear latent dynamics of this dataset is given by,

$$\frac{d\theta_t}{dt} = \omega_t, \tag{38}$$

$$\frac{d\omega_t}{dt} = -\frac{g}{l}\sin(\theta_t) - \frac{\gamma}{m}\omega_t, \tag{39}$$

where θ_t and ω_t are the angle and angular velocity, respectively, and $g = 9.81$, $l = 1$, $m = 1$, and $\gamma = 0.25$ are the acceleration due to gravity, the length of the massless cord of the pendulum, the mass of the pendulum bob, and the damping coefficient, respectively. The initial angle, θ_0, and angular velocity, ω_0, of the pendulum are chosen at random, as follows,

$$\theta_0 = \pi + \text{clip}\left(\epsilon, -2, 2\right), \tag{40}$$

$$\omega_0 = 4 \times \text{clip}\left(\epsilon, -2, 2\right), \tag{41}$$

where $\epsilon \sim \mathcal{N}(0, 1)$ and $\text{clip}(x, a, b)$ denotes clipping the value of x between a and b. The observations are Cartesian coordinates of the pendulum's bob with additive Gaussian noise, $\mathcal{N}(0, 0.05^2)$. We used the RK4 integrator to simulate the latent dynamics with a step size of 0.1s. The training, validation, and test datasets consist of 5000, 1000, and 1000 sequences of length 15s each, respectively.

CMU Motion Capture (Walking). The CMU Motion Capture database[1] comprises time series of joint angles of human subjects performing everyday activities, e.g., walking, running, and dancing. We used walking sequences of subject 35 from this database for our experiments. A preprocessed version of this dataset from Yildiz et al. (2019) consists of 23 50-dimensional sequences of 300 timesteps each, split into 16 training, 3 validation and 4 test sequences.

USHCN Climate Indicators. The USHCN Climate dataset[2] consists of measurements of five climate indicators — precipitation, snowfall, snow depth, minimum temperature, and maximum temperature — across the United States. The preprocessed version of this dataset from De Brouwer et al. (2019) contains sporadic time series from 1,114 meteorological stations with a total of 386,068 unique observations over 4 years, between 1996 and 2000. The timestamps are scaled to lie in $[0, 200]$. The 1,114 stations are split into 5 folds of 70% training, 20% validation, and 10% test stations, respectively.

Pymunk Physical Environments. The Pymunk physical environments datasets are video datasets of physical environments simulated using the Pymunk Physics engine. We used two environments proposed in Fraccaro et al. (2017): Box and Pong. Each frame of these videos is a 32 × 32 binary image. The Box dataset consists of videos of a ball moving inside a 2-dimensional box with perfectly elastic collisions with the walls of the box. The Pong dataset consists of videos of a Pong-like environment with a ball and two paddles that move to keep the ball inside the frame. Both datasets consist of 5000 training, 100 validation, and 1000 test videos with 60 frames each. We refer the reader to Fraccaro et al. (2017) for further details on how these datasets are generated[3].

C.2. Training and Evaluation Setups

Bouncing Ball and Damped Pendulum. We trained all the models on the first 10s/5s of the sequences (i.e., 100/50 steps) from the training dataset for the bouncing ball/damped pendulum datasets. We randomly dropped 30%, 50%, and 80% of the training steps for the missing-data experiments. For evaluation, we report the MSE over the missing (for imputation) and the next 200/100 timesteps (for forecast) for the bouncing ball/damped pendulum test datasets. The MSE was averaged over 5 independent runs for 50 sample trajectories.

[1] The original CMU MoCap database is available at: http://mocap.cs.cmu.edu.

[2] The original USHCN Climate dataset is available at: https://cdiac.ess-dive.lbl.gov/ftp/ushcn_daily/.

[3] The scripts for generating Pymunk datasets are available at: https://github.com/simonkamronn/kvae.

CMU Motion Capture (Walking). For Setup 1, we trained NCDSSM models on complete 300-timestep sequences from the training set. During test time, we evaluated the predictive performance on the next 297 steps with a context of the first 3 steps from the test set. For Setup 2, we trained the models on the first 200 timesteps from sequences in the training set. During test time, we provided the models with a context of the first 100 timesteps from sequences in the test set and evaluated their performance on the next 200 timesteps. We report the MSE averaged over 50 sample trajectories together with 95% prediction interval based on the t-statistic for a single run, as reported in prior works.

USHCN Climate Indicators. We trained NCDSSM models under the same setup as De Brouwer et al. (2019) using 4 years of observations from the training stations. During test time, we provided the models with the first 3 years of observations from the test set as context and evaluated their performance on the accuracy of the next 3 measurements. The MSE was computed between the mean of 50 sample forecast trajectories (simulating a point forecast) and the ground truth, averaged over the 5 folds.

Pymunk Physical Environments. We trained the models on the first 20 frames of the videos from the training dataset with 20% of the frames randomly dropped. During test time, we provided the models with a context of 20 frames and evaluated the forecast performance on the next 40 frames. We report the EMD between the predicted and the ground truth frames, averaged over 16 sample trajectories. The EMD was computed using the `ot.emd2` function from the Python Optimal Transport (POT) library (Flamary et al., 2021) with the `euclidean` metric as the cost function.

C.3. Experiment Configurations

We ran all our experiments on 2 machines with 1 Tesla T4 GPU, 16 CPUs, and 64 GB of memory each. In this section, we report training and hyperparameter configurations used in our experiments.

We optimized all models using the Adam optimizer with a learning rate of 0.01 for all the datasets except Pymunk physical environments where we used 0.002. We reduced the learning rate exponentially with a decay rate of 0.9 every 500 steps for the bouncing ball, damped pendulum, and CMU MoCap (walking) datasets, every 100 steps for the USHCN climate dataset, and every 3000 steps for the Pymunk physical environments datasets. We trained the models for 5K, 2K, 2.5K, 150, and 100K steps with a batch size of 50, 64, 16, 100, and 32 for the bouncing ball, damped pendulum, CMU MoCap (walking), USHCN climate indicators, and Pymunk physical environments, respectively.

For NCDSSM models, we used the following auxiliary inference and emission networks for each dataset:

- **Bouncing Ball, Damped Pendulum, and USHCN Climate Indicators**
  - Auxiliary inference network: `Identity()`
  - Emission network: `Identity()`
- **CMU Motion Capture (Walking)**
  - Auxiliary inference network: `Input(d)` → `Linear(64)` → `Softplus()` → `Linear (2×h)`
  - Emission network: `Input(h)` → `2×[Linear(30)` → `Softplus()]` → `Linear (d)`
- **Pymunk Physical Environments**
  - Auxiliary inference network: `Input(1, 32, 32)` → `ZeroPad2d(padding=[0, 1, 0, 1])` → `Conv2d(1, 32, kernel_size=3, stride=2)` → `ReLU()` → `2×[ZeroPad2d(padding=[0, 1, 0, 1])` → `Conv2d(32, 32, kernel_size=3, stride=2)` → `ReLU()]` → `Flatten` → `Linear(64)` → `Linear(2×h)`
  - Emission network: `Input(h)` → `Linear(512)` → `3×[Conv2d(32, 128, kernel_size=3, stride=1, padding=1)` → `ReLU()` → `PixelShuffle(upscale_factor=2)]` → `Conv2d(32, 1, kernel_size=1, stride=1)`

To ensure good initial estimation of auxiliary variables, we did not update the underlying SSM parameters for the first 100 and 1000 training steps for the CMU MoCap (walking) and Pymunk physical environments datasets, respectively. In the following, we list specific experiment configurations for individual experiments.

C.3.1. LatentODE

We used the RK4 ODE solver to integrate the encoder and drift ODEs with a step size of 0.05 for all datasets.

- **Bouncing Ball**
  - Dimension of latent state: 6

  - Dimension of observations: 1
  - Encoder network: ODEGRU with a `GRUCell(hidden_units=10)` and ODE drift function `Input(10)` → `Linear(30)` → `Tanh()` → `Linear(10)`
  - Decoder network: `Input(6)` → `Linear(10)` → `Softplus()` → `Linear(1)`
  - ODE drift function: `Input(6)` → `Linear(64)` → `Softplus()` → `Linear(6)`
- **Damped Pendulum**
  - Dimension of latent state: 6
  - Dimension of observations: 2
  - Encoder network: ODEGRU with a `GRUCell(hidden_units=10)` and ODE drift function `Input(10)` → `Linear(64)` → `Tanh()` → `Linear(10)`
  - Decoder network: `Input(6)` → `Linear(64)` → `Tanh()` → `Linear(2)`
  - ODE drift function: `Input(6)` → `Linear(64)` → `Tanh()` → `Linear(6)`
- **CMU Motion Capture (Walking)**
  - Dimension of latent state: 10
  - Dimension of observations: 50
  - Encoder network: ODEGRU with a `GRUCell(hidden_units=30)` and ODE drift function `Input(30)` → `Linear(64)` → `Tanh()` → `Linear(30)`
  - Decoder network: `Input(10)` → `2×[Linear(30)` → `Softplus()]` → `Linear(50)`
  - ODE drift function: `Input(10)` → `Linear(30)` → `Softplus()` → `Linear(10)`
- **Pymunk Physical Environments**
  - Dimension of latent state: 10
  - Dimension of observations: 1024
  - Encoder network: Same CNN encoder base as in the auxiliary inference network in NCDSSM models and ODEGRU with a `GRUCell(hidden_units=64)` and ODE drift function `Input(64)` → `Linear(64)` → `Tanh()` → `Linear(64)`
  - Decoder network: Same CNN decoder as in the emission network in NCDSSM models
  - ODE drift function: `Input(10)` → `Linear(64)` → `Tanh()` → `Linear(10)`

C.3.2. LatentSDE

For LatentSDE experiments, we additionally annealed the KL term in the objective function with a linear annealing schedule from 0 to 1 over 500 steps for all datasets except Pymunk physical environments for which we annealed over 1000 steps. As proposed in Li et al. (2020), we also provided the posterior SDEs with an additional context vector from the encoder to incorporate information from later observations. We used the RK4 ODE solver to integrate the encoder ODEs and the Euler-Maruyama SDE solver to integrate the prior/posterior SDEs with a step size of 0.05 for all datasets.

- **Bouncing Ball**
  - Dimension of latent state: 6
  - Dimension of context vector: 3
  - Dimension of observations: 1
  - Encoder network: ODEGRU with a `GRUCell(hidden_units=10)` and ODE drift function `Input(10)` → `Linear(64)` → `Tanh()` → `Linear(10)`
  - Decoder network: `Input(6)` → `Linear(64)` → `Softplus()` → `Linear(1)`
  - Posterior SDE drift function: `Input(6+3)` → `Linear(64)` → `Softplus()` → `Linear(6)`
  - Prior SDE drift function: `Input(6)` → `Linear(64)` → `Softplus()` → `Linear(6)`
  - Posterior/Prior SDE diffusion function: `6×[Input(1)` → `Linear(64)` → `Softplus()` → `Linear(1)]`
- **Damped Pendulum**
  - Dimension of latent state: 6
  - Dimension of context vector: 3
  - Dimension of observations: 2
  - Encoder network: ODEGRU with a `GRUCell(hidden_units=10)` and ODE drift function `Input(10)` → `Linear(64)` → `Tanh()` → `Linear(10)`
  - Decoder network: `Input(6)` → `Linear(64)` → `Tanh()` → `Linear(2)`
  - Posterior SDE drift function: `Input(6+3)` → `Linear(64)` → `Softplus()` → `Linear(6)`
  - Prior SDE drift function: `Input(6)` → `Linear(64)` → `Softplus()` → `Linear(6)`

  - Posterior/Prior SDE diffusion function: 6×[Input(1) → Linear(64) → Softplus() → Linear(1)]
- **CMU Motion Capture (Walking)**
  - Dimension of latent state: 10
  - Dimension of context vector: 3
  - Dimension of observations: 50
  - Encoder network: ODEGRU with a GRUCell(hidden_units=30) and ODE drift function Input(30) → Linear(64) → Tanh() → Linear(30)
  - Decoder network: Input(10) → 2×[Linear(30) → Softplus()] → Linear(50)
  - Posterior SDE drift function: Input(10+3) → Linear(30) → Softplus() → Linear(10)
  - Prior SDE drift function: Input(10) → Linear(30) → Softplus() → Linear(10)
  - Posterior/Prior SDE diffusion function: 10×[Input(1) → Linear(30) → Softplus() → Linear(1)]
- **Pymunk Physical Environments**
  - Dimension of latent state: 10
  - Dimension of context vector: 4
  - Dimension of observations: 1024
  - Encoder network: Same CNN encoder base as in the auxiliary inference network in NCDSSM models and ODEGRU with a GRUCell(hidden_units=64) and ODE drift function Input(64) → Linear(64) → Tanh() → Linear(64)
  - Decoder network: Same CNN decoder as in the emission network in NCDSSM models
  - Posterior SDE drift function: Input(10+4) → Linear(64) → Tanh() → Linear(10) → Tanh()
  - Prior SDE drift function: Input(10) → Linear(64) → Tanh() → Linear(10) → Tanh()
  - Posterior/Prior SDE diffusion function: 10×[Input(1) → Linear(64) → Softplus() → Linear(1)]

C.3.3. NCDSSM-LTI

- **Bouncing Ball**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 1
  - Dimension of observations (d): 1
  - Integrator: Analytic
- **Damped Pendulum**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 2
  - Dimension of observations (d): 2
  - Integrator: Analytic
- **CMU Motion Capture (Walking)**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 6
  - Dimension of observations (d): 50
  - Integrator: Analytic
- **USHCN Climate Indicators**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 5
  - Dimension of observations (d): 5
  - Integrator: Euler with step size 0.1
- **Pymunk Physical Environments**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 4
  - Dimension of observations (d): 1024
  - Integrator: RK4 with step size 0.05

C.3.4. NCDSSM-NL

We set the diffusion function to $\mathbf{G}(\cdot, t) = \mathbf{I}$ for all datasets.

- **Bouncing Ball**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 1
  - Dimension of observations (d): 1
  - Drift function ($\mathbf{f}$): `Input(m)` $\rightarrow$ `Linear(64)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(m)`
  - Integrator: RK4 with step size 0.05
- **Damped Pendulum**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 2
  - Dimension of observations (d): 2
  - Drift function ($\mathbf{f}$): `Input(m)` $\rightarrow$ `Linear(64)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(m)`
  - Integrator: RK4 with step size 0.05
- **CMU Motion Capture (Walking)**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 6
  - Dimension of observations (d): 50
  - Drift function ($\mathbf{f}$): `Input(m) -> SN(Linear(30)) -> Softplus() -> SN(Linear(m))`
  - Integrator: RK4 with step size 0.05
- **USHCN Climate Indicators**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 5
  - Dimension of observations (d): 5
  - Drift function ($\mathbf{f}$): `Input(m)` $\rightarrow$ `SN(Linear(64))` $\rightarrow$ `Softplus()` $\rightarrow$ `SN(Linear(m))`
  - Integrator: Euler with step size 0.1
- **Pymunk Physical Environments**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 4
  - Dimension of observations (d): 1024
  - Drift function ($\mathbf{f}$): `Input(m)` $\rightarrow$ `Linear(64)` $\rightarrow$ `Tanh()` $\rightarrow$ `Linear(m)` $\rightarrow$ `Tanh()`
  - Integrator: RK4 with step size 0.05

C.3.5. NCDSSM-LL

We set the α-network to `Input(m)` $\rightarrow$ `Linear(64)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(K)` for all datasets.

- **Bouncing Ball**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 1
  - Dimension of observations (d): 1
  - Number of base matrices (K): 5
  - Integrator: RK4 with step size 0.05
- **Damped Pendulum**
  - Dimension of state (m): 6
  - Dimension of auxiliary variables (h): 2
  - Dimension of observations (d): 2
  - Number of base matrices (K): 5
  - Integrator: RK4 with step size 0.05
- **CMU Motion Capture (Walking)**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 6
  - Dimension of observations (d): 50
  - Integrator: RK4 with step size 0.05
- **USHCN Climate Indicators**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 5

  - Dimension of observations (d): 5
  - Number of base matrices (K): 10
  - Integrator: Euler with step size 0.1
- **Pymunk Physical Environments**
  - Dimension of state (m): 10
  - Dimension of auxiliary variables (h): 4
  - Dimension of observations (d): 1024
  - Number of base matrices (K): 10
  - Integrator: RK4 with step size 0.05

D. Additional Results

Table 5 shows the number of trainable parameters in each model for different experiments. NCDSSM models obtain better performance on every dataset with significantly fewer parameters. Table 6 shows the goodness-of-fit coefficient (R^2) for ordinary least squares regression with the latent states as features, and the ground truth angle and angular velocity as targets. NCDSSM-NL and NCDSSM-LL models obtain a high R^2 coefficient showing that the latent states learned by these models are informative about the true latent state (angle and angular velocity).

Figs. 4 and 5 show sample predictions from the *best run* of each model for different missing data settings on the bouncing ball and the damped pendulum datasets, respectively. For the bouncing ball experiment, both LatentODE and LatentSDE learn that the dataset exhibits a zig-zag pattern but are unable to accurately extrapolate it beyond the training context. In the case of damped pendulum, LatentODE and LatentSDE perform well on the low missing data settings (0% and 30%) but completely fail on the more challenging settings of 50% and 80% missing data. In contrast, NCDSSM-NL and NCDSSM-LL generate accurate predictions across datasets and missing data settings. Furthermore, while the predictions shown in Figs. 4 and 5 are from the best performing runs of each model, they represent a typical run for NCDSSM-NL and NCDSSM-LL. On the other hand, the prediction quality from LatentODE and LatentSDE models varies significantly across random initializations.

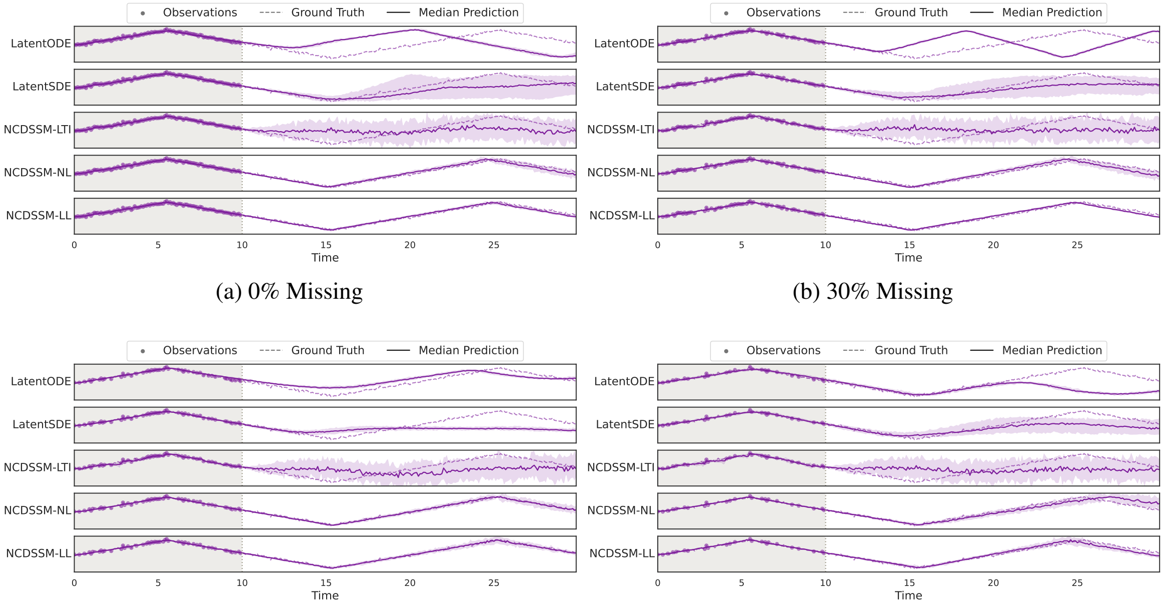
Fig. 6 shows the variation of the EMD with time for different models on the box and pong datasets. All models have EMD close to 0 in the context window from 0-2s; however, in the forecast horizon from 2-6s, the EMD rises gradually for NCDSSM-NL and NCDSSM-LL but rapidly and irregularly for other models. Figs. 7 and 8 show sample predictions from different models on the box and the pong datasets, respectively. NCDSSM-NL and NCDSSM-LL generate accurate predictions whereas LatentODE and LatentSDE perform significantly worse.

Table 5. The number of trainable parameters in every model for different experiments.

Model	Number of Parameters				
	Bouncing Ball	Damped Pendulum	MoCap Walking (Setup 2)	USHCN	Pymunk Environments
LatentODE	2094	3336	15454	–	204243
LatentSDE	5461	5557	17187	–	208043
NCDSSM-LTI	63	72	11080	185	165911
NCDSSM-NL	859	862	11620	1439	167165
NCDSSM-LL	974	977	12509	2439	168165

Table 6. Goodness-of-fit coefficient (R^2) of ordinary least squares (OLS) regression for the *best run* of each model on the Pendulum dataset. The latent states are treated as features and ground truth angle — transformed into polar coordinates: $\sin(\text{angle})/\cos(\text{angle})$ — and angular velocity as targets.

Model	$\sin(\text{angle})/\cos(\text{angle})$ R^2 (↑) (% Missing)				Angular Velocity R^2 (↑) (% Missing)			
	0%	30%	50%	80%	0%	30%	50%	80%
LatentODE	0.000 / 0.802	0.000 / 0.735	0.000 / 0.744	0.000 / 0.626	0.001	0.000	0.000	0.000
LatentSDE	0.953 / 0.960	0.918 / 0.957	0.000 / 0.817	0.000 / 0.513	0.970	0.962	0.001	0.000
NCDSSM-LTI	0.593 / 0.537	0.604 / 0.468	0.477 / 0.796	0.481 / 0.705	0.349	0.388	0.162	0.305
NCDSSM-NL	0.984 / **0.990**	0.982 / 0.985	**0.973** / 0.976	**0.905** / **0.920**	**0.986**	0.969	0.935	**0.859**
NCDSSM-LL	**0.986** / 0.989	**0.983** / **0.989**	0.972 / **0.980**	0.875 / 0.888	0.972	**0.978**	**0.955**	0.827



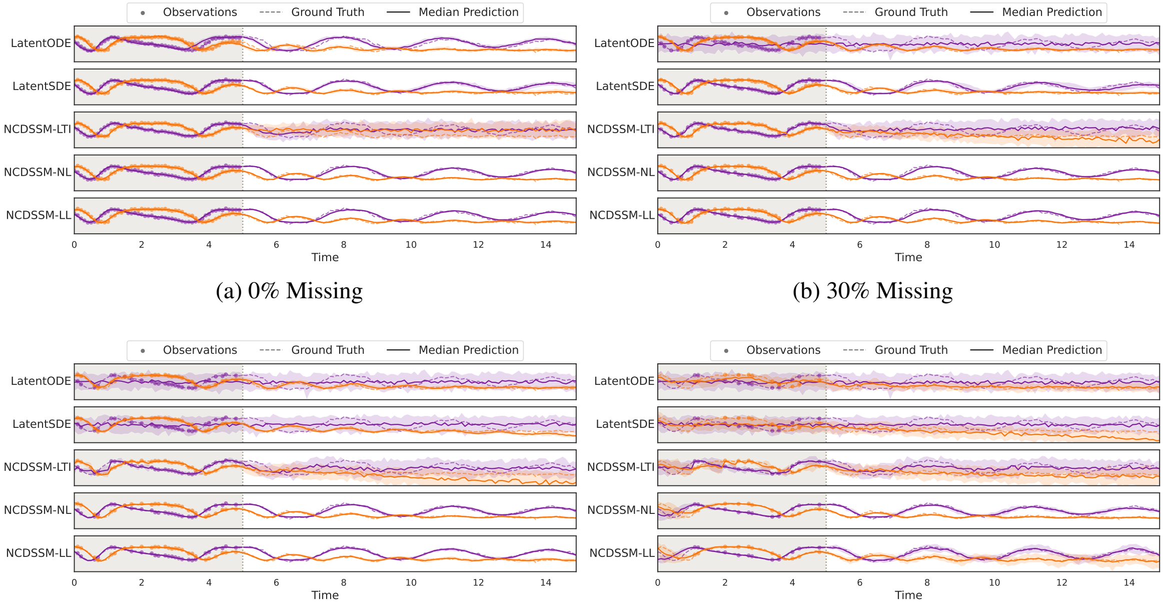
(a) 0% Missing

(b) 30% Missing

(c) 50% Missing

(d) 80% Missing

Figure 4. Predictions from different models on the bouncing ball dataset for the 0%, 30%, 50%, and 80% missing data settings. The ground truth is shown using dashed lines with observed points in the context window (gray shaded region) shown as filled circles. The vertical dashed gray line marks the beginning of the forecast horizon. Solid lines indicate median predictions with 90% prediction intervals shaded around them.



(a) 0% Missing

(b) 30% Missing

(c) 50% Missing

(d) 80% Missing

Figure 5. Predictions from different models on the damped pendulum dataset for the 0%, 30%, 50%, and 80% missing data settings. The ground truth is shown using dashed lines with observed points in the context window (gray shaded region) shown as filled circles. The vertical dashed gray line marks the beginning of the forecast horizon. Solid lines indicate median predictions with 90% prediction intervals shaded around them. The purple and orange colors indicate observation dimensions.

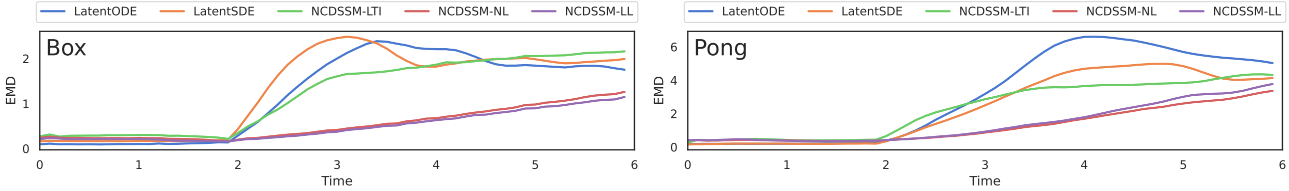
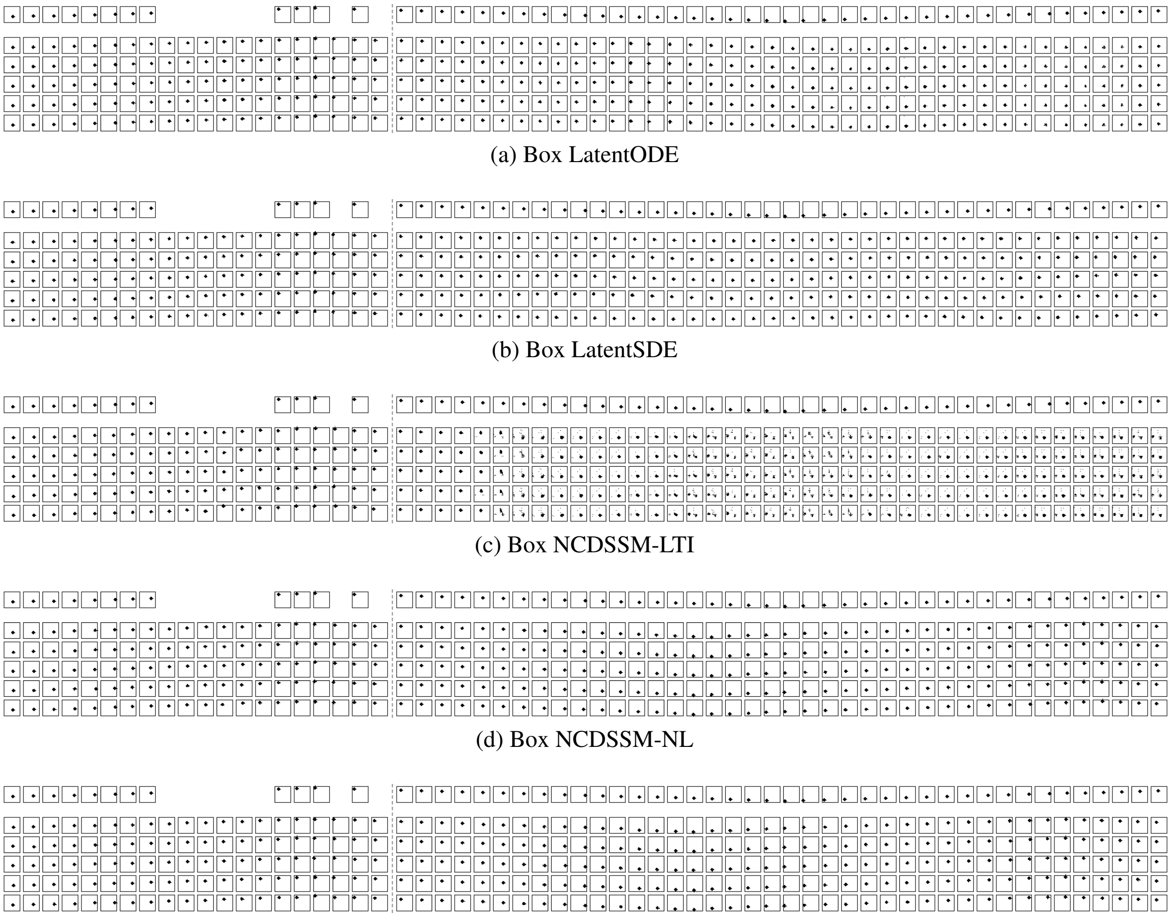


Figure 6. Variation of EMD over time for the Box (left) and Pong (right) datasets. The EMD rises gradually with time for NCDSSM-LL and NCDSSM-NL but rapidly and irregularly for other models.



(a) Box LatentODE

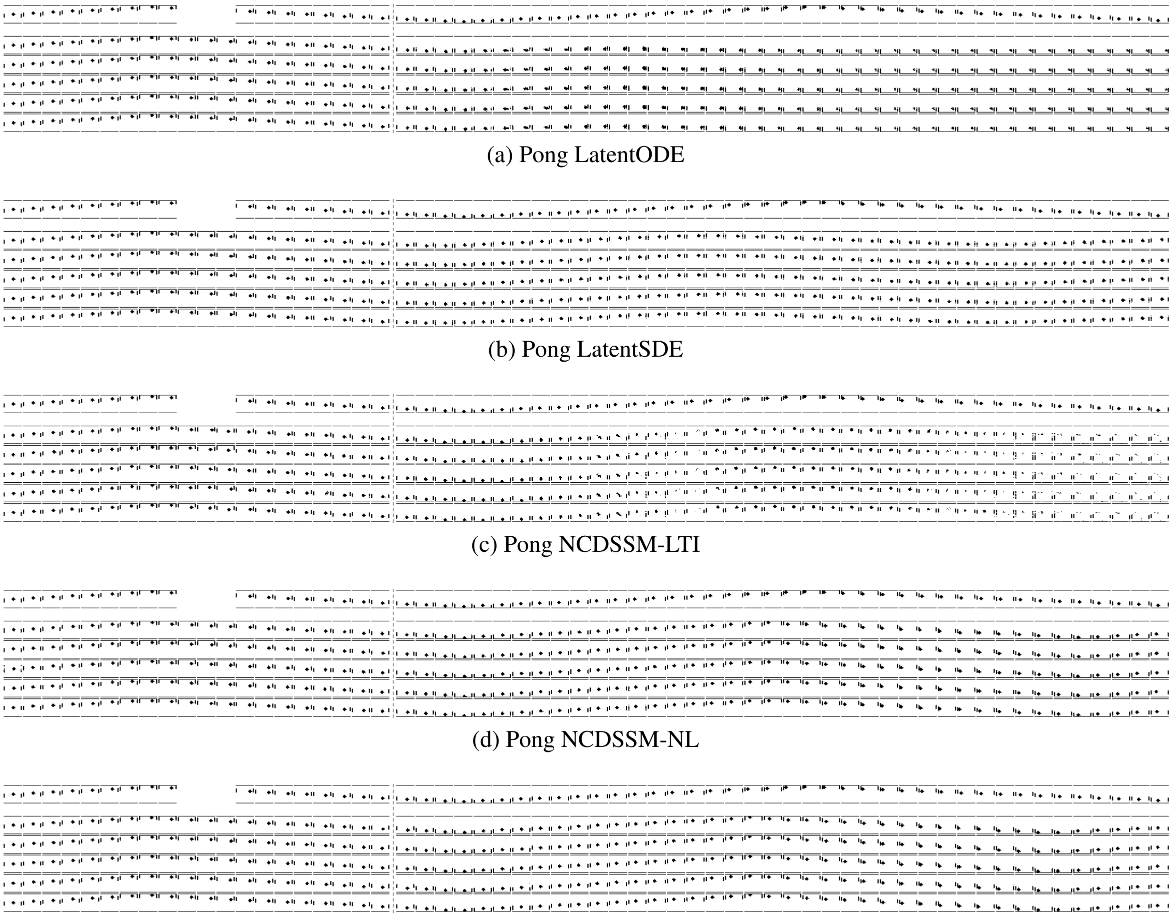
(b) Box LatentSDE

(c) Box NCDSSM-LTI

(d) Box NCDSSM-NL

(e) Box NCDSSM-LL

Figure 7. Sample predictions from different models on the Box dataset. The top row in each figure is the ground truth with some missing observations in the context window (before the dashed grey line). The next five rows show trajectories sampled from each model. Best viewed zoomed-in on a computer.



(a) Pong LatentODE

(b) Pong LatentSDE

(c) Pong NCDSSM-LTI

(d) Pong NCDSSM-NL

(e) Pong NCDSSM-LL

Figure 8. Sample predictions from different models on the Pong dataset. The top row in each figure is the ground truth with some missing observations in the context window (before the dashed grey line). The next five rows show trajectories sampled from each model. Best viewed zoomed-in on a computer.