Neural Stochastic Flows: Solver-Free Modelling and Inference for SDE Solutions

Naoki Kiyohara[1,2*] **Edward Johns**[1] **Yingzhen Li**[1]
[1]Imperial College London [2]Canon Inc.
{n.kiyohara23, e.johns, yingzhen.li}@imperial.ac.uk

Abstract

Stochastic differential equations (SDEs) are well suited to modelling noisy and/or irregularly-sampled time series, which are omnipresent in finance, physics, and machine learning applications. Traditional approaches require costly simulation of numerical solvers when sampling between arbitrary time points. We introduce *Neural Stochastic Flows* (NSFs) and their latent dynamic versions, which learn (latent) SDE transition laws directly using conditional normalising flows, with architectural constraints that preserve properties inherited from stochastic flow. This enables sampling between arbitrary states in a single step, providing up to two orders of magnitude speedup for distant time points. Experiments on synthetic SDE simulations and real-world tracking and video data demonstrate that NSF maintains distributional accuracy comparable to numerical approaches while dramatically reducing computation for arbitrary time-point sampling, enabling applications where numerical solvers remain prohibitively expensive.

1 Introduction

Stochastic differential equations (SDEs) underpin models in finance, physics, biology, and modern machine learning systems [39, 50]: they capture how a state $\boldsymbol{x}_t \in \mathbb{R}^d$ evolves following a velocity field whilst being perturbed by random noise. In many real-time settings such as robots, trading algorithms, or digital twins, one often requires the *transition law* $p(\boldsymbol{x}_t \mid \boldsymbol{x}_s)$ describing the probability distribution of future states given earlier states over arbitrary time gaps $t - s$ [30]. Conventional approaches handle this transition law by learning neural (latent) SDEs [33] from data and simulating numerical solvers with many small steps [19, 30], which incurs high computational costs. In this regard, neural flow [4] techniques for ordinary differential equations (ODEs) bypass numerical integration by directly learning the flow map with architectural constraints and regularisation terms. However, these methods inherently cannot express stochastic dynamics required for SDE modelling. In the domain of diffusion models [12, 20, 47], a line of work leverages the associated *probability flow ODE* (PF-ODE): earlier distillation approaches compress iterative samplers by matching teacher-student trajectories under the PF-ODE [44], while consistency-style methods learn direct mappings between time points [48], with trajectory-level variants also proposed [26]. However, these techniques are tied to particular boundary conditions and diffusion processes.

An alternative approach for modelling stochastic dynamics data is through stochastic flows [32], which, under suitable conditions, describe families of strong solutions to SDEs via mappings that evolve initial states over time under shared stochasticity. These flows naturally define the transition law $p(\boldsymbol{x}_t \mid \boldsymbol{x}_s)$, and when parameterised by neural networks, enable direct learning of efficient one-step sampling between arbitrary time points. However, it remains an unsolved challenge for such network design, due to the requirements of satisfying several properties, e.g., identity mapping when

[*]Project page: https://nkiyohara.github.io/nsf-neurips2025/

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

 https://doi.org/10.52202/085713-0571

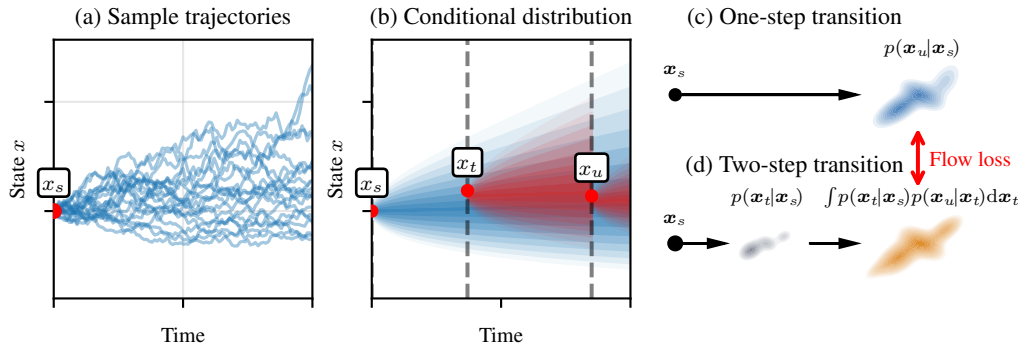


Figure 1: Comparison between (a) traditional neural SDE methods requiring numerical integration and (b) our NSF, where blue and red contours represent conditional distributions for one-step sampling and recursive application, respectively. Panels (c) and (d) illustrate our flow loss concept, ensuring distributional consistency between one-step and two-step transitions through intermediate states.

$t = s$, Markov property, and Chapman–Kolmogorov flow property that ensures self-consistency of transition laws with different number of steps.

Contributions. We introduce *Neural Stochastic Flows* (NSFs) to address the network design challenge of parameterising stochastic flows. NSFs employ conditional normalising flows [41, 52] to directly learn the SDE transition distributions $p(\boldsymbol{x}_t \mid \boldsymbol{x}_s)$ for any $s < t$, where the normalising flow architecture design ensures the validity of identity, Markov, and (for autonomous SDEs) stationarity properties. Specifically, given the present state $\boldsymbol{x}_s$ and elapsed time Δt, NSF draws a single Gaussian noise vector and transforms it through specially designed affine coupling layers [13, 28] that reduce to identity maps when $\Delta t \coloneqq t - s = 0$ [4]. These transformations have parameters conditioned on $(\boldsymbol{x}_s, \Delta t, s)$ and scale their effect with the time interval. A bi-directional KL divergence based regularisation loss is designed to further encourage the network to satisfy Chapman–Kolmogorov flow property. Since all transformations are bijective, the transition log-density is available in closed form, allowing both training and inference to proceed without SDE solvers. A side-by-side schematic of solver-based vs. NSF sampling is shown in Fig. 1.

To summarise, Neural Stochastic Flows can:

- Learn an SDE's weak solution, in the form of a conditional distribution, directly for solver-free training and inference, with architectural constraints and loss function design to enforce desirable stochastic flow properties;
- Enable efficient one-step sampling between arbitrary time points within the trained horizon (thus suitable for modelling irregularly sampled time series), with maximum gains for distant time points;
- Be extended to noisy and/or partially observed data scenarios through latent dynamic modelling, in similar fashions as variational state-space model and latent SDEs [31, 33].

Empirically, across diverse benchmarks such as stochastic Lorenz attractor, CMU Motion Capture, and Stochastic Moving MNIST, our approach maintains distributional accuracy comparable to or better than numerical solver methods, while delivering up to two orders of magnitude faster predictions over arbitrary time intervals, with the largest gains on long-interval forecasts.

2 Background

Stochastic Differential Equations. Stochastic differential equations (SDEs) model dynamical systems subject to random perturbations:

$$\mathrm{d}\boldsymbol{x}_t = \boldsymbol{\mu}(\boldsymbol{x}_t, t)\,\mathrm{d}t + \boldsymbol{\sigma}(\boldsymbol{x}_t, t)\,\mathrm{d}\boldsymbol{W}_t, \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the state, $\boldsymbol{\mu} : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ is the drift, $\boldsymbol{\sigma} : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^{n\times m}$ is the diffusion, and $\boldsymbol{W}_t$ is an m-dimensional Wiener process. Neural SDEs parameterise these terms with neural networks, enabling learning from data. In many applications, the conditional probability distribution

$p(\boldsymbol{x}_t \mid \boldsymbol{x}_s)$ for $s < t$ is crucial for uncertainty quantification and probabilistic forecasting. However, while numerical solvers can generate sample trajectories, they do not provide direct access to this distribution and incur computational costs that scale with the time gap $t - s$.

Stochastic Flows of Diffeomorphisms. An alternative perspective on SDEs comes from the theory of stochastic flows, which characterises the solution space through time-dependent mappings. For an Itô SDE as in Eq. (1) with smooth coefficients, these solutions form a *stochastic flow of diffeomorphisms* [32]. This is defined by a measurable map $\boldsymbol{\phi}_{s,t} : \mathbb{R}^n \times \Omega \to \mathbb{R}^n$ where $0 < s < t < +\infty$ and Ω is the sample space, satisfying:

1. **Diffeomorphism**: For any $s < t$ and $\boldsymbol{\omega} \in \Omega$, the map $\boldsymbol{x} \mapsto \boldsymbol{\phi}_{s,t}(\boldsymbol{x}, \boldsymbol{\omega})$ is almost surely a diffeomorphism.
2. **Independence**: Maps $\boldsymbol{\phi}_{t_1,t_2}, \ldots, \boldsymbol{\phi}_{t_{n-1},t_n}$ for any non-overlapping sequence $0 \leq t_1 \leq t_2 \leq \ldots \leq t_n$ are independent, which yields the Markov property of the flow.
3. **Flow property**: For any $0 \leq t_1 \leq t_2 \leq t_3$, any point $\boldsymbol{x} \in \mathbb{R}^n$, and any $\boldsymbol{\omega} \in \Omega$:
$$\boldsymbol{\phi}_{t_1,t_3}(\boldsymbol{x}, \boldsymbol{\omega}) = \boldsymbol{\phi}_{t_2,t_3}\big(\boldsymbol{\phi}_{t_1,t_2}(\boldsymbol{x}, \boldsymbol{\omega}), \boldsymbol{\omega}\big)\,.$$
4. **Identity property**: For any $t \geq 0$, any point $\boldsymbol{x} \in \mathbb{R}^n$, and any $\boldsymbol{\omega} \in \Omega$: $\boldsymbol{\phi}_{t,t}(\boldsymbol{x}, \boldsymbol{\omega}) = \boldsymbol{x}$.

Furthermore, when the SDE is autonomous, where the drift and diffusion terms are independent of time, an additional property holds:

5. **Stationarity**: For any $s \leq t$, $r \geq 0$, and $\boldsymbol{\omega} \in \Omega$, the maps $\boldsymbol{\phi}_{s,t}(\boldsymbol{x}, \boldsymbol{\omega})$ and $\boldsymbol{\phi}_{s+r,t+r}(\boldsymbol{x}, \boldsymbol{\omega})$ have the same probability distribution.

Solver-based neural SDEs (also see Section 5) require costly numerical integration. Leveraging the flow properties formalised above, in the next section we introduce a solver-free alternative, NSFs, as efficient models for stochastic systems that provide direct access to transition probability distributions of SDE-governed processes.

3 Neural Stochastic Flows

While stochastic flows of diffeomorphisms provide a powerful theoretical framework for understanding strong solutions of SDEs, they require modelling infinite-dimensional sample paths, which is computationally intractable. Instead, in this section we focus on developing neural network architectures for modelling the probability distributions that characterise weak solutions of SDEs, by deriving appropriate conditions and loss functions for the network using properties of stochastic flows.

For an SDE, the relationship between a strong solution (represented by a stochastic flow $\boldsymbol{\phi}$) to its corresponding weak solution (represented by a conditional probability distribution) is expressed as:

$$p(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i) \coloneqq \int_{\Omega} \delta\left(\boldsymbol{x}_{t_j} - \boldsymbol{\phi}_{t_i,t_j}(\boldsymbol{x}_{t_i}, \boldsymbol{\omega})\right) p(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}, \tag{2}$$

where $\delta(\cdot)$ denotes the Dirac delta function and Ω represents the infinite-dimensional sample space containing complete Brownian motion trajectories. Instead of directly modelling the map $\boldsymbol{\phi}$, we approximate the resulting probability distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i) \approx p(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i)$ using a *Neural Stochastic Flow* (NSF), a parametric model $\boldsymbol{f}_{\boldsymbol{\theta}}$ that transforms Gaussian samples $\boldsymbol{\varepsilon}$ into the desired distributions via conditional normalising flows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i) = \int \delta\big(\boldsymbol{x}_{t_j} - \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_i}, t_i, t_j - t_i, \boldsymbol{\varepsilon})\big)\,\mathcal{N}(\boldsymbol{\varepsilon} \mid \mathbf{0}, \boldsymbol{I})\,\mathrm{d}\boldsymbol{\varepsilon}, \tag{3}$$

Therefore $\boldsymbol{f}_{\boldsymbol{\theta}}$ serves as our parametric model to approximate the SDE's weak solution.

Conditions. Under the NSF framework, in below we reformulate the conditions of stochastic flows as strong solutions of SDEs to conditions of NSFs as weak solutions:

1. **Independence**: For any sequence $0 \leq t_1 \leq t_2 \leq \ldots \leq t_n$, the conditional probabilities $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_2} \mid \boldsymbol{x}_{t_1}; t_1, t_2 - t_1), \ldots, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_n} \mid \boldsymbol{x}_{t_{n-1}}; t_{n-1}, t_n - t_{n-1})$ must be independent.
2. **Flow property**: For any $0 \leq t_i \leq t_j \leq t_k$, the joint distribution must satisfy the Chapman–Kolmogorov equation:
$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}; t_i, t_k - t_i) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}; t_j, t_k - t_j) p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i)\,\mathrm{d}\boldsymbol{x}_{t_j}. \tag{4}$$

https://doi.org/10.52202/085713-0571

3. **Identity property**: At the same time t, the distribution must reduce to a delta function, indicating no change: $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_i} = \boldsymbol{x} \mid \boldsymbol{x}_{t_i}; t_i, 0) = \delta(\boldsymbol{x} - \boldsymbol{x}_{t_i})$.

For autonomous SDEs, we restate the stationarity condition in terms of our parametric model:

4. **Stationarity**: The conditional distributions must exhibit stationarity such that for any $t_i, t_j, r \geq 0$, $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}; t_i, t_j - t_i) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j+r} \mid \boldsymbol{x}; t_i + r, t_j - t_i)$.

3.1 Conditional Normalising Flow Design

Let $\boldsymbol{c} \coloneqq (\boldsymbol{x}_{t_i}, \Delta t, t_i)$ denote the conditioning parameters, where t_i is included only for non-autonomous systems and omitted otherwise, and let $\Delta t \coloneqq t_j - t_i$. We instantiate the sampling procedure of the flow distribution (Eq. (3)) as

$$\boldsymbol{z} = \underbrace{\boldsymbol{x}_{t_i} + \Delta t \cdot \mathrm{MLP}_{\mu}(\boldsymbol{c}; \boldsymbol{\theta}_{\mu})}_{\boldsymbol{\mu}(\boldsymbol{c})} + \underbrace{\sqrt{\Delta t} \cdot \mathrm{MLP}_{\sigma}(\boldsymbol{c}; \boldsymbol{\theta}_{\sigma})}_{\boldsymbol{\sigma}(\boldsymbol{c})} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{5}$$

$$\boldsymbol{x}_{t_j} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{c}) = \boldsymbol{f}_L(\cdot; \boldsymbol{c}, \boldsymbol{\theta}_L) \circ \boldsymbol{f}_{L-1}(\cdot; \boldsymbol{c}, \boldsymbol{\theta}_{L-1}) \circ \cdots \circ \boldsymbol{f}_1(\boldsymbol{z}; \boldsymbol{c}, \boldsymbol{\theta}_1). \tag{6}$$

Our architecture integrates a parametric Gaussian initialisation with a sequence of bijective transformations. The state-dependent Gaussian, centred at $\boldsymbol{\mu}(\boldsymbol{c})$ with noise scale $\boldsymbol{\sigma}(\boldsymbol{c})$, follows the similar form to the Euler–Maruyama discretisation with drift scaled by Δt and diffusion by $\sqrt{\Delta t}$. The subsequent bijective transformations $\boldsymbol{f}_1$ through $\boldsymbol{f}_L$ are implemented as conditioned coupling flows [4, 13, 28] whose parameters depend on $\boldsymbol{c}$. Each layer splits the state $\boldsymbol{z}$ into two partitions $(\boldsymbol{z}_{\mathrm{A}}, \boldsymbol{z}_{\mathrm{B}})$ and applies an affine update to one partition conditioned on the other and on $\boldsymbol{c}$:

$$\boldsymbol{f}_i(\boldsymbol{z}; \boldsymbol{c}, \boldsymbol{\theta}_i) = \mathrm{Concat}\Big(\boldsymbol{z}_{\mathrm{A}}, \boldsymbol{z}_{\mathrm{B}} \odot \exp\big(\Delta t\, \mathrm{MLP}^{(i)}_{\text{scale}}(\boldsymbol{z}_{\mathrm{A}}, \boldsymbol{c}; \boldsymbol{\theta}^{(i)}_{\text{scale}})\big) + \Delta t\, \mathrm{MLP}^{(i)}_{\text{shift}}(\boldsymbol{z}_{\mathrm{A}}, \boldsymbol{c}; \boldsymbol{\theta}^{(i)}_{\text{shift}})\Big), \tag{7}$$

with alternating partitions across layers. The explicit Δt factor ensures $\boldsymbol{f}_i(\boldsymbol{z}; \boldsymbol{c}, \boldsymbol{\theta}_i) = \boldsymbol{z}$ when $\Delta t = 0$ and, combined with the form of the base Gaussian (Eq. (5)), preserves the identity at zero time gap. Stacking such layers yields an expressive diffeomorphism [49] while keeping the Jacobian log-determinant tractable for loss computation. Independence property is ensured by the conditional sampling on the initial state $\boldsymbol{x}_{t_i}$ without any overlap between the transitions, and stationarity is obtained by omitting t_i from $\boldsymbol{c}$ when modelling autonomous SDEs.

3.2 A Regularisation Loss for Flow Property

To train an NSF, apart from using supervised learning loss functions based on e.g., maximum likelihood, we add an additional discrepancy term $\mathcal{L}_{\text{flow}}$ that encourages the flow property as formalised in Eq. (4). We require a measure of mismatch between the one-step distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})$ and the two-step marginal $\int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j}$. Several candidates exist such as optimal-transport/Wasserstein, Stein, adversarial, and kernel MMD. Among them, we choose KL divergences because we can minimise the upper-bounds of the KL divergences in a tractable way, and since KL is asymmetric, we use both directions: the forward KL promotes coverage, while the reverse KL penalises unsupported regions. Direct KLs remain intractable due to the marginalisation over intermediate state $\boldsymbol{x}_{t_j}$ in two-step side. Therefore, we introduce a bridge distribution $b_{\boldsymbol{\xi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})$ as an auxiliary variational distribution [1, 40, 45] and optimise variational upper bounds for both directions (Appendix B).

Specifically, for the forward KL divergence:

$$\begin{aligned}
&D_{\mathrm{KL}}\left(p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}\right) \,\middle\|\, \int p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}\right) p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}\right) \mathrm{d}\boldsymbol{x}_{t_j}\right) \\
&\leq \mathop{\mathbb{E}}_{\boldsymbol{x}_{t_k} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}_{t_i})} \left[\mathop{\mathbb{E}}_{\boldsymbol{x}_{t_j} \sim b_{\boldsymbol{\xi}}(\cdot \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})} \left[\log \frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}\right)\, b_{\boldsymbol{\xi}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k}\right)}{p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}\right)\, p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}\right)} \right] \right] \\
&=: \mathcal{L}_{\text{flow, 1-to-2}}\left(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k\right).
\end{aligned} \tag{8}$$

And for the reverse KL divergence:

$$D_{\mathrm{KL}}\left(\int p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}\right) p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}\right) \mathrm{d}\boldsymbol{x}_{t_j} \,\middle\|\, p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}\right)\right)$$

Algorithm 1: Optimisation procedure for (Latent) Neural Stochastic Flows
Follow ♦ for NSF, ♠ for Latent NSF; lines without markers are run for both models.

Input: Transition dataset $\mathcal{D}$; learning rates η_θ, η_ξ (and η_ϕ, η_ψ for latent); inner steps K
Output: Trained parameters $\boldsymbol{\theta}, \boldsymbol{\xi}$ (and $\boldsymbol{\phi}, \boldsymbol{\psi}$ for latent)

while *not converged* **do**
  Sample a minibatch from $\mathcal{D}$
  ♠ Encode sampled data into latent states using Eq. (15)
  Sample time triplets (t_i, t_j, t_k) with $t_i < t_j < t_k$ (see Appendix B for details)
  for $k = 1$ **to** K **do**
    Compute $\mathcal{L}_{\text{flow}}(\boldsymbol{\theta}, \boldsymbol{\xi})$ using Eq. (10)
    Update bridge parameters: $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi} - \eta_\xi \nabla_{\boldsymbol{\xi}} \mathcal{L}_{\text{flow}}$
  ♦ Compute total loss $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\xi})$ using Eq. (11)
  ♠ Compute total loss $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\phi}, \boldsymbol{\psi})$ using Eq. (18)
  ♦ Update NSF parameters: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_\theta \nabla_{\boldsymbol{\theta}} \mathcal{L}$
  ♠ Update NSF/encoder/decoder parameters: $\{\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}\} \leftarrow \{\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}\} - \eta_{\{\theta,\phi,\psi\}} \nabla_{\{\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\psi}\}} \mathcal{L}$

$$
\begin{aligned}
&\leq \mathop{\mathbb{E}}_{\boldsymbol{x}_{t_j} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}_{t_i})} \left[\mathop{\mathbb{E}}_{\boldsymbol{x}_{t_k} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x}_{t_j})} \left[\log \frac{p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}\right) \, p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}\right)}{b_{\boldsymbol{\xi}}\left(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k}\right) \, p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}\right)} \right] \right] \\
&=: \mathcal{L}_{\text{flow, 2-to-1}}\left(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k\right). \qquad (9)
\end{aligned}
$$

The flow loss $\mathcal{L}_{\text{flow}}$ combines the above two terms, balancing the consistency in both directions:

$$
\mathcal{L}_{\text{flow}}(\boldsymbol{\theta}, \boldsymbol{\xi}) = \mathop{\mathbb{E}}_{p(t_i, t_j, t_k)} \left[\mathcal{L}_{\text{flow, 1-to-2}}(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k) + \mathcal{L}_{\text{flow, 2-to-1}}(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k) \right], \qquad (10)
$$

The total loss for training NSFs combines the negative log-likelihood objective with the flow loss:

$$
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\xi}) = - \mathop{\mathbb{E}}_{(\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j}, t_i, t_j) \sim \mathcal{D}} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, t_j - t_i) \right] + \lambda \, \mathcal{L}_{\text{flow}}(\boldsymbol{\theta}, \boldsymbol{\xi}), \qquad (11)
$$

where $\mathcal{D}$ represents the dataset and λ is a hyperparameter that controls the strength of the flow consistency constraint. The model parameters $\boldsymbol{\theta}$ and bridge model parameters $\boldsymbol{\xi}$ are optimised using gradient-based methods; a procedure is summarised in Algorithm 1 (see Appendix C.1 for details).

4 Latent Neural Stochastic Flows

Irregularly-sampled real-world sequences seldom expose the full system state; instead we observe noisy measurements $\boldsymbol{o}_{t_i}$ at times $0 = t_0 < t_1 < \cdots < t_T$. To model the hidden continuous-time dynamics while remaining solver-free, we introduce the *Latent Neural Stochastic Flows* (Latent NSFs) as variational state-space models (VSSMs) whose transition kernels are governed by NSFs.

4.1 Recap: Variational State-Space Models (VSSMs)

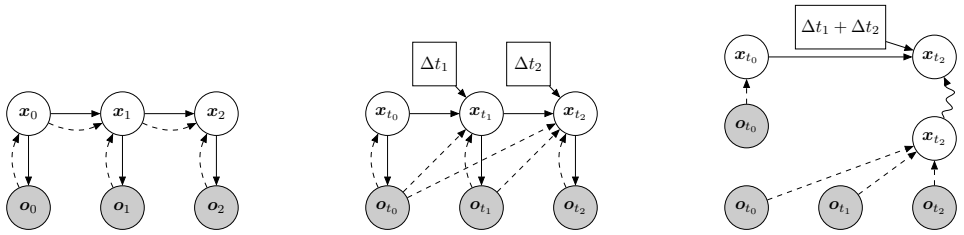
A VSSM [8, 21, 22, 31] extends the variational autoencoder framework [29, 42] to sequential data by endowing a sequence of latent states $\boldsymbol{x}_{0:T}$ with Markovian dynamics while explaining the observations $\boldsymbol{o}_{0:T}$ through a conditional emission process. The joint distribution factorises as

$$
p_{\boldsymbol{\theta},\boldsymbol{\psi}}(\boldsymbol{x}_{0:T}, \boldsymbol{o}_{0:T}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) p_{\boldsymbol{\psi}}(\boldsymbol{o}_t \mid \boldsymbol{x}_t), \qquad (12)
$$

where each conditional is typically Gaussian with its distributional parameters produced by neural networks. Amortised inference is carried out using a Gaussian encoder $q_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{o}_t)$ for every time-step t, and the encoder and decoder are jointly trained using the negative evidence lower bound (NELBO) loss, also known as the variational free energy (VFE):

$$
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\psi}, \boldsymbol{\phi}) = \mathop{\mathbb{E}}_{q_{\boldsymbol{\phi}}} \left[-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) - \sum_{t=1}^{T} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) + \log p_{\boldsymbol{\psi}}(\boldsymbol{o}_t \mid \boldsymbol{x}_t) - \log q_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{o}_t) \right] \right]. \qquad (13)
$$

https://doi.org/10.52202/085713-0571



(a) Variational State-Space Model (b) Latent Neural Stochastic Flow (c) Skip KL Divergence

Figure 2: Graphical models. (a) Standard state-space model with discrete-time transitions between states. (b) NSF model with continuous-time transitions parameterised by time intervals ($\Delta t_i \coloneqq t_i - t_{i-1}$). (c) Example of skip KL divergence over two time steps. The upper row shows the two-step-ahead prediction from the posterior at t_0 via NSF, while the lower row shows the posterior at t_2 conditioned on all observations $(\boldsymbol{o}_{t_0}, \boldsymbol{o}_{t_1}, \boldsymbol{o}_{t_2})$. Across all models: solid arrows represent generative processes, dashed arrows represent inference processes, wavy arrows in (c) represent KL divergences.

4.2 Neural Stochastic Flows as Latent Transition Kernels

To model irregularly-sampled time series, we require the latent dynamics to evolve in continuous-time. We therefore implement the transition distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})$ in (12) with an NSF $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_{i-1}, \Delta t_i)$ which parameterises the weak solution of an SDE as follows. As illustrated in Fig. 2(a), a standard VSSM uses discrete-time transitions, whereas latent NSFs in Fig. 2(b) incorporates continuous-time dynamics through time intervals.

Generative model. Let $\boldsymbol{x}_{t_i} \in \mathbb{R}^d$ be the latent state and $\Delta t_i \coloneqq t_i - t_{i-1}$. The joint distribution factorises as

$$p_{\boldsymbol{\theta},\boldsymbol{\psi}}\big(\boldsymbol{x}_{t_{0:T}}, \boldsymbol{o}_{t_{0:T}}\big) = p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_0}) \prod_{i=1}^{T} p_{\boldsymbol{\theta}}\big(\boldsymbol{x}_{t_i} \mid \boldsymbol{x}_{t_{i-1}}; t_{i-1}, \Delta t_i\big)\, p_{\boldsymbol{\psi}}(\boldsymbol{o}_{t_i} \mid \boldsymbol{x}_{t_i})\,, \tag{14}$$

where $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_i} \mid \boldsymbol{x}_{t_{i-1}}; \cdot)$ is an NSF, hence supports arbitrary Δt_i without numerical integration.

Variational posterior. For the inference model, we employ a recurrent neural network encoder that processes the observation sequence [7]:

$$q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_{0:T}} \mid \boldsymbol{o}_{\le t_T}) = \prod_{i=0}^{T} \mathcal{N}\big(\boldsymbol{x}_{t_i} \mid \boldsymbol{m}_{t_i}, \operatorname{diag}\big(\boldsymbol{s}_{t_i}^2\big)\big)\,, \quad (\boldsymbol{m}_{t_i}, \boldsymbol{s}_{t_i}) = \mathrm{GRU}_{\boldsymbol{\phi}}\big([\boldsymbol{o}_{t_i}, \Delta t_i, t_i], \boldsymbol{h}_{t_{i-1}}\big)\,, \tag{15}$$

with $\boldsymbol{h}_{t_{-1}} = \mathbf{0}$, and the absolute time (t_{i-1} in Eq. (14) and t_i in Eq. (15)) is only included for non-autonomous systems and omitted when modelling autonomous SDEs.

Learning objective. Our goal is to train generative model $p_{\boldsymbol{\theta},\boldsymbol{\psi}}\big(\boldsymbol{x}_{t_{0:T}}, \boldsymbol{o}_{t_{0:T}}\big)$ and inference model $q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_{0:T}} \mid \boldsymbol{o}_{\le t_T})$. A standard choice is the β-weighted negative ELBO (β-NELBO) loss [18]:

$$\mathcal{L}_{\beta\text{-NELBO}} = \sum_{i=0}^{T} \Big[- \mathop{\mathbb{E}}_{q_{\boldsymbol{\phi}}} \big[\log p_{\boldsymbol{\psi}}(\boldsymbol{o}_{t_i} \,|\, \boldsymbol{x}_{t_i})\big] + \beta D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_i} \,|\, \boldsymbol{o}_{\le t_i}) \,\big\|\, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_i} \,|\, \boldsymbol{x}_{t_{i-1}})\big)\Big], \tag{16}$$

which focuses only on adjacent time steps, potentially leading to the error accumulation for long-term dependencies. To address this, we propose to add a skip-ahead KL divergence loss (Fig. 2(c)):

$$\mathcal{L}_{\text{skip}} = \sum_{i=0}^{T-\tau} \mathop{\mathbb{E}}_{j \sim \mathcal{U}\{i+2, \tau\}} \big[D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \,|\, \boldsymbol{o}_{\le t_j}) \,\big\|\, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \,|\, \boldsymbol{x}_{t_i})\big)\big]. \tag{17}$$

Unlike traditional overshooting methods [16] that require recursive transitions, latent NSF enables direct sampling across arbitrary time gaps, enabling this objective to be computed efficiently.

Combining the above two losses and the flow loss (Eq. (10)), we get the total loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\beta\text{-NELBO}} + \lambda\, \mathcal{L}_{\text{flow}} + \beta_{\text{skip}} \mathcal{L}_{\text{skip}}, \tag{18}$$

where λ and β_{skip} are hyperparameters that control the strength of the flow loss and skip-ahead KL divergence loss, respectively.

A training procedure for the latent model is summarised in Algorithm 1 (see Appendix C.2).

5 Related Work

We categorise prior work in terms of whether learning and/or sampling avoid fine-grained numerical time-stepping, and the class of dynamics targeted, as summarised in Table 1.

Modelling general ODEs. Neural ODEs [6] learn a parametric vector field that is integrated by a numerical solver at both training and test time. Their runtime therefore scales with the number of function evaluations required by the solver. Neural flows [4] side-step this cost by learning the solution map directly.

Table 1: Solver requirement across continuous-time differential equation models. 'Pre-defined diffusion SDEs/ODEs' refers to a boundary-conditioned diffusion process bridging a fixed base distribution and the data distribution utilised in diffusion models.

Method(s)	Target dynamics	Solver-free training	Solver-free inference
Neural ODEs [6]	General ODEs	✗	✗
Neural flows [4]	General ODEs	✓	✓
Score-based diffusion via reverse SDEs/PF-ODEs [47]; flow matching [34]	Pre-defined diffusion SDEs/ODEs	✓	✗
Progressive distillation [44]	Pre-defined diffusion SDEs/ODEs	✗	✓
Consistency models [26, 48]; rectified flows [35]	Pre-defined diffusion SDEs/ODEs	✓	✓
Neural (latent) SDEs [25, 33, 38, 46, 50]	General Itô SDEs	✗	✗
ARCTA [9]; SDE matching [3]	General Itô SDEs	✓	✗
Neural Stochastic Flows	General Itô SDEs	✓	✓

Modelling prescribed SDEs/PF-ODEs. Score-based generative models [47] and flow matching [34] are continuous-time generative models that learn reverse SDEs/PF-ODEs of boundary-conditioned diffusion processes. Fast generation methods have been developed through learning direct mappings [26, 48] or vector-field straightening [34, 53]. However, they are specifically designed for prescribed boundary-conditioned diffusion processes, and do not provide a general transition density for arbitrary SDEs.

Modelling general SDEs. Neural (latent) SDEs [25, 33, 38, 50] utilise neural networks to model the drift and diffusion terms of an SDE and approximate its trajectories via stochastic solvers, whose computational cost grows linearly with the prediction horizon. Recent advances have aimed to mitigate this cost by improving sampling efficiency [46] or by introducing solver-free approaches such as ARCTA [9] and SDE matching [3], which learn latent SDEs by modelling posterior marginals and aligning them to the prior dynamics. However, all these methods remain solver-dependent at inference time.

Position of the present work. NSF unifies the solver-free philosophy of neural flows with the expressive power of neural (latent) SDEs. It learns the Markov transition density as a conditional normalising flow, which satisfies the conditions of weak solutions of SDEs by design and regularisation objectives. Unlike diffusion model-specific accelerations, NSF handles arbitrary Itô SDEs and extends naturally to latent sequence models. The resulting method removes numerical integration while retaining closed-form likelihoods.

6 Experiments

We evaluate NSF and latent NSF on three diverse tasks: modelling a synthetic stochastic Lorenz attractor [33, 36], predicting real-world human motion (CMU Motion Capture [15]), and generative video modelling (Stochastic Moving MNIST [11]). Our goal is to show that (latent) NSF matches or improves upon the accuracy (task-specific metrics) of solver-based neural SDEs while drastically reducing computational cost in terms of FLOPs and runtime. Full details are in Appendix E.

6.1 Stochastic Lorenz Attractor

We first test NSF on the stochastic Lorenz system [33, 36], a standard chaotic dynamics benchmark. We use the setup from Li et al. [33], generating 1,024 training trajectories and comparing against baselines including latent SDE [33], SDE matching [3], and Stable Neural SDE variants (Neural LSDE, Neural GSDE, Neural LNSDE) [38] as baselines. Performance is measured by KL divergence (estimated via kernel density estimation) between generated and true distributions, and computational cost via FLOPs and runtime (details in Appendix E.2).

https://doi.org/10.52202/085713-0571

Table 2: Comparison on KL divergence and FLOPs at different time steps. H_{pred} indicates the maximum single-step prediction horizon for NSF; see text for details on recursive application. Average runtime per 100 samples for latent SDE (JAX): 124–148 ms; NSF (JAX): 0.3 ms (see Appendix E.2.5).

Method	$t = 0.25$		$t = 0.5$		$t = 0.75$		$t = 1.0$	
	KL	kFLOPs	KL	kFLOPs	KL	kFLOPs	KL	kFLOPs
Latent SDE [33]	2.1 ± 0.9	959	1.8 ± 0.1	1,917	0.9 ± 0.3	2,839	1.5 ± 0.5	3,760
Neural LSDE [38]	1.3 ± 0.4	1,712	7.2 ± 1.4	3,416	74.5 ± 24.6	5,057	53.1 ± 29.3	6,699
Neural GSDE [38]	1.2 ± 0.4	1,925	3.9 ± 0.3	3,848	20.2 ± 7.6	5,698	14.1 ± 8.4	7,548
Neural LNSDE [38]	1.7 ± 0.4	1,925	4.6 ± 0.7	3,848	57.3 ± 16.4	5,698	44.6 ± 23.4	7,548
SDE matching [3]								
$\Delta t = 0.0001$	4.3 ± 0.7	184,394	5.3 ± 1.0	368,787	3.4 ± 0.8	553,034	3.8 ± 1.0	737,354
$\Delta t = 0.01$	6.3 ± 0.4	1,917	11.7 ± 0.5	3,834	7.9 ± 0.3	5,677	6.0 ± 0.3	7,520
NSF (ours)								
$H_{\text{pred}} = 1.0$	0.8 ± 0.7	53	1.3 ± 0.1	53	0.6 ± 0.3	53	0.2 ± 0.6	53
$H_{\text{pred}} = 0.5$	2.4 ± 1.9	53	1.3 ± 0.1	53	1.0 ± 0.4	105	1.7 ± 1.1	105
$H_{\text{pred}} = 0.25$	1.2 ± 0.7	53	1.2 ± 0.1	105	0.8 ± 0.4	156	1.3 ± 0.8	208

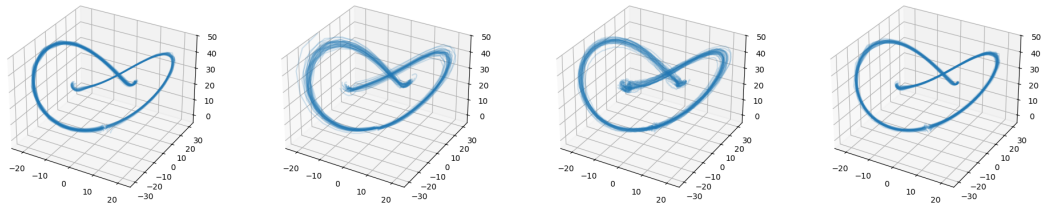
Fig. 3 shows paths from each method; NSF traces visually coincide with the true attractor whereas solver-based methods overspread. Quantitatively in Table 2, NSF ($H_{\text{pred}} = 1.0$, single-step) achieves the lowest KL divergence and with significantly fewer FLOPs. This leads to substantial runtime savings (approx. 0.3 ms vs. $>$ 100 ms per batch, Appendix E.2.5). Notably, our flow-based training objective enables both single-step and recursive application variants to maintain relatively low KL divergence across different time horizons, with NSF remaining computationally cheaper in these configurations. This confirms NSF's capability to accurately model complex SDEs efficiently.

Table 3: Test MSE and 95% confidence interval based on t-statistic on Motion Capture datasets. † indicates results from Yildiz et al. [54], * from Li et al. [33], ‡ from Course and Nair [9], and § from Ansari et al. [2]. All other results are our reproductions. Average runtime per 100 samples: latent SDE (JAX) - 75ms; Latent NSF (JAX) - 3.5ms (detailed in Appendix E.3.7).

Methods	Setup 1	Setup 2
npODE [17]	$22.96^{\dagger}$	–
Neural ODE [6]	$22.49 \pm 0.88^{\dagger}$	–
ODE2VAE-KL [54]	$8.09 \pm 1.95^{\dagger}$	–
Latent ODE [43]	$5.98 \pm 0.28^{*}$	$31.62 \pm 0.05^{\S}$
Latent SDE [33]	$12.91 \pm 2.90^{\text{I}}$	$9.52 \pm 0.21^{\S}$
Latent Approx SDE [46]	$7.55 \pm 0.05^{\S}$	10.50 ± 0.86
ARCTA [9]	$7.62 \pm 0.93^{\ddagger}$	9.92 ± 1.82
NCDSSM [2]	$5.69 \pm 0.01^{\S}$	$4.74 \pm 0.01^{\S}$
SDE matching [3]	$\mathbf{5.20 \pm 0.43}^{2}$	4.26 ± 0.35
Latent NSF (ours)	8.62 ± 0.32	$\mathbf{3.41 \pm 0.27}$

6.2 CMU Motion Capture

We evaluate Latent NSF on the CMU Motion Capture walking dataset [15]. The 23-sequence walking subset is down-sampled to 300 time steps and split 16/3/4 for train/validation/test, matching prior work [51, 54]. We report two established protocols.



(a) Ground truth (b) Latent SDE (c) SDE matching (d) NSF (one-step, ours)

Figure 3: Comparison of generated samples (64 samples per panel) on the stochastic Lorenz attractor. Baseline methods are simulated step-by-step. For NSF, each point is an independent sample from the learnt conditional distribution $p(\boldsymbol{x}_t \mid \boldsymbol{x}_s)$ originating from the same initial state and seed, connected visually for comparison. This visualisation choice aids in assessing distributional accuracy over time.

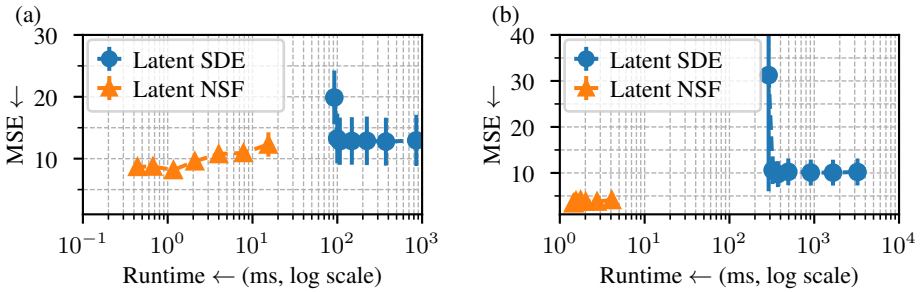


Figure 4: Trade-off between prediction accuracy (MSE) and computational cost (runtime, using JAX) for latent SDE and Latent NSF on CMU Motion Capture dataset. (a) Setup 1: Within-horizon forecasting. (b) Setup 2: Beyond-horizon extrapolation.

Within-horizon forecasting (Setup 1) [33, 54]. All 300 time steps are used during training; for tests, only the first three observations are revealed and the model must forecast the remaining 297 steps.

Beyond-horizon extrapolation (Setup 2) [2]. The last third segment of every sequence (steps 200–299) is withheld from training. At test time the model receives the first 100 observations and must predict the next 200 steps that lie beyond the training horizon.

Table 3 shows the results. In Setup 1, Latent NSF (single-step prediction) performs similarly to latent SDE models. Crucially, in the Setup 2 extrapolation task, latent NSF achieves state-of-the-art MSE.

These extrapolation gains align with the relevance of state-dependent stochasticity in human motion. Nonlinear SDE formulations often require learning complex, time-varying posterior structures (e.g., controlled SDEs or complicated conditional marginals). By contrast, Latent NSF directly models the Markov transition with conditional normalising flows, sidesteps these optimisation challenges.

Fig. 4 illustrates the trade-off between prediction accuracy and computational cost. While latent SDE adjusts the discretisation step size to balance accuracy and speed, Latent NSF controls this trade-off by varying the number of recursive applications. Latent NSF achieves better performance while being approximately more than two orders of magnitude faster than latent SDE in both setups, as detailed in Appendix E.3.7.

6.3 Stochastic Moving MNIST

Finally, we test Latent NSF on high-dimensional video using a Stochastic Moving MNIST [11] variant with physically plausible bouncing dynamics: digits undergo perfect specular reflection with small angular noise, avoiding the unphysical velocity resets of the original code. The task involves modelling two digits moving in a 64x64 frame. We train on 60k sequences and test on 10k held-out sequences. We evaluate using Fréchet distances (FD) on embeddings from a pre-trained SRVP model [14], measuring static content, dynamics, and frame similarity (validated in Appendix G). We compare our Latent NSF model against the latent SDE [33] as the main baseline, representing the current representative solver-based neural SDE model.

Table 4 and Fig. 5 show that Latent NSF (recursive and one-step) achieves competitive Static FD and Frame-wise FD compared to the latent SDE baseline. Dynamics FD for recursive NSF is slightly higher in this run, but visualisations suggest comparable dynamics capture (see Appendix E.4.5). The decline after step 35 across all models reflects the digits approaching a uniform spatial distribution, indicating a limitation of this metric when evaluating long-term predictions in feature spaces fitted to Gaussian distributions. Nevertheless, Latent NSF offers significant potential for computational speedup over the solver-based latent SDE (e.g., latent SDE: 751 ms vs. Latent NSF (one-step): 358

[1]Li et al. [33] report 4.03 $\pm$ 0.20 for Setup 1; our re-implementation and other reproductions observe higher MSE. Details are provided in Appendix E.3.2.

[2]Bartosh et al. [3] report 4.50 $\pm$ 0.32 for Setup 1; however, at the time of writing, official code for CMU Motion Capture is not yet available. We report our re-implementation here to maintain consistency across Setups 1 and 2. Details of our reproduction are provided in Appendix E.3.3.

https://doi.org/10.52202/085713-0571

Table 4: Fréchet distance comparison on Stochastic Moving MNIST using pre-trained SRVP embeddings. The lower the better. Note that dynamics FD is ill-defined for one-step simulation of latent NSF, which only predicts the marginal distribution of the future frames conditioned on the last observation.

Methods	Static FD	Dynamics FD	Frame-wise FD
Latent SDE	2.66 ± 0.87	5.39 ± 3.10	0.58 ± 0.21
Latent NSF (recursive)	2.36 ± 0.60	7.76 ± 2.56	0.63 ± 0.17
Latent NSF (one-step)	1.67 ± 0.47	–	0.63 ± 0.15

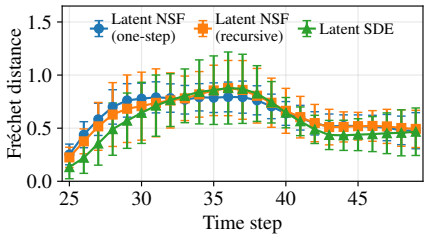


Figure 5: Comparison of frame-wise Fréchet distance across time-steps on Stochastic Moving MNIST.

ms for full-sequence prediction; see Appendix E.4.5). This demonstrates NSF's promise for scaling to high-dimensional stochastic sequence modelling.

7 Discussion

Neural Stochastic Flows (NSFs) represent a novel paradigm for continuous-time stochastic modelling by directly learning weak solutions to SDEs as conditional distributions. Through carefully designed architectural constraints and regularisation objectives, NSF circumvents numerical integration while providing closed-form transition densities via normalising flows. The latent NSF extension enables applications to partially observed or high-dimensional time series, preserving efficient one-step transitions within a principled variational state-space modelling framework.

Our comprehensive evaluation across chaotic dynamical systems, human motion capture, and video sequences demonstrates that the proposed flow-based approach achieves comparable or superior performance relative to solver-based neural SDEs while reducing computational requirements by two orders of magnitude. Moreover, it attains state-of-the-art long-horizon extrapolation performance in latent settings. These results collectively establish that learning distributional flows, rather than the underlying vector fields required by numerical solvers, constitutes an effective and computationally efficient paradigm for real-time stochastic modelling.

Limitations. Several limitations remain. First, Chapman–Kolmogorov consistency is enforced only approximately via variational bounds, potentially yielding discrepancies beyond the training regime. We recommend systematic monitoring of flow losses on held-out data with appropriate adjustments. Second, the selection of maximum one-shot horizon necessitates careful balance between computational efficiency and predictive accuracy. Third, while our affine coupling architecture guarantees analytical Jacobian computation, it imposes constraints on the form of the architecture.

Future work. Extending NSF to action-conditioned settings could enable direct integration with control algorithms such as model predictive control and reinforcement learning. Additionally, bridging NSF with diffusion models presents opportunities for leveraging complementary strengths of both frameworks in continuous-time modelling. Another natural direction is to explore stronger flow parameterisations. In particular, transformer-based flows such as TarFlow [55] may relax the structural constraints of affine coupling while retaining tractable Jacobians, potentially further improving expressivity. These extensions would significantly broaden the applicability of NSF to decision-making and generative tasks.

Application-wise, the resulting analytical tractability, coupled with empirically demonstrated two-order-of-magnitude computational speedups, renders NSF particularly suitable for diverse applications, from high-frequency trading scenarios requiring real-time robotic control and fast simulations to large-scale digital twin implementations.

Broader Impact Statement. This paper presents work whose goal is to advance machine learning research. There may exist potential societal consequences of our work, however, none of which we feel must be specifically highlighted here at the moment of paper publication.

References

[1] F. V. Agakov and D. Barber. An auxiliary variational method. In *Neural Information Processing, 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004, Proceedings*, volume 3316 of *Lecture Notes in Computer Science*, pages 561–566. Springer, 2004. doi: 10.1007/978-3-540-30499-9_86. URL https://doi.org/10.1007/978-3-540-30499-9_86.

[2] A. F. Ansari, A. Heng, A. Lim, and H. Soh. Neural continuous-discrete state space models for irregularly-sampled time series. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 926–951. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/ansari23a.html.

[3] G. Bartosh, D. Vetrov, and C. A. Naesseth. SDE matching: Scalable and simulation-free training of latent stochastic differential equations. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[4] M. Biloš, J. Sommer, S. S. Rangapuram, T. Januschowski, and S. Günnemann. Neural flows: Efficient alternative to neural odes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[6] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 6572–6583, 2018.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/V1/D14-1179. URL https://doi.org/10.3115/v1/d14-1179.

[8] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Advances in Neural Information Processing Systems*, 2015.

[9] K. Course and P. B. Nair. Amortized reparametrization: Efficient and scalable variational inference for latent SDEs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5yZiP9fZNv.

[10] R. Daems, M. Opper, G. Crevecoeur, and T. Birdal. Variational inference for SDEs driven by fractional noise. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=rtx8B94JMS.

[11] E. Denton and R. Fergus. Stochastic video generation with a learned prior. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1182–1191. PMLR, 2018. URL http://proceedings.mlr.press/v80/denton18a.html.

[12] P. Dhariwal and A. Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 8780–8794, 2021.

[13] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[14] J. Franceschi, E. Delasalles, M. Chen, S. Lamprier, and P. Gallinari. Stochastic latent residual video prediction. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3233–3246. PMLR, 2020. URL http://proceedings.mlr.press/v119/franceschi20a.html.

[15] Z. Gan, C. Li, R. Henao, D. E. Carlson, and L. Carin. Deep temporal sigmoid belief networks for sequence modeling. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/95151403b0db4f75bfd8da0b393af853-Paper.pdf.

[16] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

[17] M. Heinonen, C. Yildiz, H. Mannerström, J. Intosalmi, and H. Lähdesmäki. Learning unknown ODE models with Gaussian processes. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1959–1968. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/heinonen18a.html.

[18] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

[19] D. J. Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Review*, 43(3):525–546, 2001. doi: 10.1137/S0036144500378302. URL https://doi.org/10.1137/S0036144500378302.

[20] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6840–6851, 2020.

[21] M. J. Johnson, D. K. Duvenaud, A. B. Wiltschko, R. P. Adams, and S. R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, 2016.

[22] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=HyTqHL5xg.

[23] P. Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021.

[24] P. Kidger and C. Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[25] P. Kidger, J. Foster, X. Li, H. Oberhauser, and T. Lyons. Neural sdes as infinite-dimensional gans. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 5541–5553, 2021.

[26] D. Kim, C. Lai, W. Liao, N. Murata, Y. Takida, T. Uesaka, Y. He, Y. Mitsufuji, and S. Ermon. Consistency trajectory models: Learning probability flow ODE trajectory of diffusion. In *International Conference on Learning Representations (ICLR)*, 2024.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[28] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 10215–10224, 2018.

[29] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

[30] P. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Stochastic Modelling and Applied Probability. Springer Berlin Heidelberg, 2011. ISBN 9783540540625. URL https://books.google.co.uk/books?id=BCvtssom1CMC.

[31] R. G. Krishnan, U. Shalit, and D. Sontag. Structured inference networks for nonlinear state space models. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2101–2109. AAAI Press, 2017.

[32] H. Kunita. *Stochastic Flows and Stochastic Differential Equations*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1990. ISBN 9780521599252. URL https://books.google.co.uk/books?id=_S1RiCosqbMC.

[33] X. Li, T. Wong, R. T. Q. Chen, and D. Duvenaud. Scalable gradients for stochastic differential equations. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[34] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

[35] X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations (ICLR)*, 2023.

[36] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2):130 – 141, 1963. doi: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2. URL https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml.

[37] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[38] Y. Oh, D. Lim, and S. Kim. Stable neural stochastic differential equations in analyzing irregular time series data. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4VIgNuQ1pY.

[39] B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications (Universitext)*. Springer, 6th edition, 2003.

[40] R. Ranganath, D. Tran, and D. M. Blei. Hierarchical variational models. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 324–333. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/ranganath16.html.

[41] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/rezende15.html.

[42] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1278–II–1286. JMLR.org, 2014.

[43] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

 https://doi.org/10.52202/085713-0571

[44] T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TIdIXIpzhoI.

[45] T. Salimans, D. P. Kingma, and M. Welling. Markov chain Monte Carlo and variational inference: Bridging the gap. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1218–1226. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/salimans15.html.

[46] A. Solin, E. Tamir, and P. Verma. Scalable inference in sdes by direct matching of the Fokker–Planck–Kolmogorov equation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[47] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[48] Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, pages 32211–32252, 2023.

[49] T. Teshima, I. Ishikawa, K. Tojo, K. Oono, M. Ikeda, and M. Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3362–3373. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/2290a7385ed77cc5592dc2153229f082-Paper.pdf.

[50] B. Tzen and M. Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

[51] J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):283–298, Feb. 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1167. URL https://doi.org/10.1109/TPAMI.2007.1167.

[52] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

[53] L. Yang, Z. Zhang, Z. Zhang, X. Liu, M. Xu, W. Zhang, C. Meng, S. Ermon, and B. Cui. Consistency flow matching: Defining straight flows with velocity consistency, 2024. URL https://arxiv.org/abs/2407.02398.

[54] C. Yildiz, M. Heinonen, and H. Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/99a401435dcb65c4008d3ad22c8cdad0-Paper.pdf.

[55] S. Zhai, R. ZHANG, P. Nakkiran, D. Berthelot, J. Gu, H. Zheng, T. Chen, M. Á. Bautista, N. Jaitly, and J. M. Susskind. Normalizing flows are capable generative models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=2uheUFcFsM.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions—our neural stochastic flow (NSF) model, latent NSF extension, and its empirical benefits—and the experimental results in Sections 6 and Appendix E directly support these claims without exaggeration.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The current manuscript contains an explicit "Limitations" paragraph in Section 7 discussing weaknesses.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

https://doi.org/10.52202/085713-0571

Answer: [NA]

Justification: The work presents an empirical method without formal theorems; no theoretical claims are made, so formal assumptions and proofs are not applicable.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Implementation details and full hyper-parameter are provided in Appendix E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: For now, we include the related references for the sources of the datasets. We will release the code and detailed running instructions upon publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Appendix E enumerate every training and evaluation hyper-parameter (learning rate, optimizer, batch size, number of epochs, etc.) for each experiment.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All reported metrics include standard deviation or 95% confidence intervals computed over five random seeds; methodology and seed information are detailed in Section 6 and Appendix E.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

https://doi.org/10.52202/085713-0571

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments were conducted on a single NVIDIA RTX 3090 GPU (24 GB); typical training times are under few days per dataset as reported in Appendix E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: In this paper, we introduce work designed to push the boundaries of machine learning. While our methods could carry ethical implications, we do not believe any require specific discussion at this point in the submission process.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: A dedicated "Broader Impact Statement" section is provided near the end of the paper, outlining potential positive and negative societal impacts.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Whenever we use any assets, we always cite the original asset source.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

 https://doi.org/10.52202/085713-0571

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We fully describe our contributions and any assets used in the paper. We will also be releasing code after the publishing of the paper.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The study involves only publicly available simulation and motion-capture datasets and does not recruit human subjects or use crowdsourcing.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: No human-subject experiments were conducted, so IRB approval is not required.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: Generative LLM tools (e.g., ChatGPT, GitHub Copilot, Claude) were used solely for auxiliary tasks such as code refactoring, generating data visualization scripts, and proofreading the manuscript. These tools did not contribute to the development of the core methodology, experimental design, or interpretation of results.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

https://doi.org/10.52202/085713-0571

Appendix Contents

 https://doi.org/10.52202/085713-0571

A Architecture

A.1 Neural Stochastic Flow

We refer to the main text for the NSF architecture and notation (Section 3, Eqs. (5) and (6) in the main text). Here we only note implementation specifics. MLP_μ and MLP_σ are neural networks that share parameters and produce split outputs for the mean and variance components. Softplus activation is used for the variance component to ensure positivity. For stability, the scale heads end with a $\tanh$ multiplied by a learnt scalar to bound log-scales [13]. Exact log-densities are computed via the change-of-variables formula; see Section D.

A.2 Bridge Model

Here, we describe the architecture of the bridge model used as auxiliary variational distribution [1, 40, 45]. The bridge model characterises the conditional distribution $b_{\boldsymbol{\xi}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j})$ for any intermediate time point t where $t_i < t < t_j$, given the boundary states. Combined with flow loss (Eq. (10)) minimisation, this enables inference of intermediate states when both endpoints are known. Let $\boldsymbol{c}_{\text{br}} \coloneqq (\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j}, \Delta t, \tau, t_i)$ denote the conditioning parameters, where $\tau \coloneqq (t - t_i)/\Delta t$ is the normalised time and $\Delta t \coloneqq t_j - t_i$ is the time interval, and the initial time t_i is ommitted for autonomous systems.

Similarly to the main NSF, we first draw a sample from a parametric Gaussian distribution:

$$\boldsymbol{z}_t = \underbrace{\boldsymbol{x}_{t_i} + \tau\big(\boldsymbol{x}_{t_j} - \boldsymbol{x}_{t_i}\big) + \alpha(\tau)\,\text{MLP}_\mu(\boldsymbol{c}_{\text{br}})}_{\boldsymbol{\mu}(t)} + \underbrace{\sqrt{\alpha(\tau)\Delta t}\,\text{Softplus}\big(\text{MLP}_\sigma(\boldsymbol{c}_{\text{br}})\big)}_{\boldsymbol{\sigma}(t)} \odot \boldsymbol{\varepsilon}, \tag{19}$$

where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a standard normal noise vector, $\alpha(\tau) \coloneqq \tau(1-\tau)$ is the standard Brownian-bridge factor, and MLP_μ and MLP_σ are neural networks that share parameters and produce split outputs for the mean and variance components. The intermediate state is then obtained by applying a flow transformation:

$$\boldsymbol{x}_t = \boldsymbol{f}_{\boldsymbol{\xi}}(\boldsymbol{z}_t, \boldsymbol{c}_{\text{br}}) = \boldsymbol{f}_L(\cdot; \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}_L) \circ \boldsymbol{f}_{L-1}(\cdot; \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}_{L-1}) \circ \cdots \circ \boldsymbol{f}_1(\boldsymbol{z}_t; \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}_1). \tag{20}$$

Note that $\alpha(\tau)$ vanishes at the endpoints ($\tau = 0$ and $\tau = 1$, corresponding to $t = t_i$ and $t = t_j$), ensuring consistency with boundary conditions. The flow transformation $\boldsymbol{f}_{\boldsymbol{\xi}}$ is implemented using a series of conditioned bijective transformations, as detailed below.

A.2.1 Conditioned Bijective Transformations

Each layer of the bridge flow follows the same coupling structure as in the main NSF (Section 3). Given an input $\boldsymbol{z}_t$, we split it into two parts $(\boldsymbol{z}_{\text{A}}, \boldsymbol{z}_{\text{B}}) = \text{Split}(\boldsymbol{z}_t)$ with alternating partitioning across layers, and apply the transformation:

$$\begin{aligned} &\boldsymbol{f}_i(\boldsymbol{z}_t; \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}_i) \\ &= \text{Concat}\Big(\boldsymbol{z}_{\text{A}},\ \boldsymbol{z}_{\text{B}} \odot \exp\big(\alpha(\tau)\,\text{MLP}^{(i)}_{\text{scale}}(\boldsymbol{z}_{\text{A}}, \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}^{(i)}_{\text{scale}})\big) + \alpha(\tau)\,\text{MLP}^{(i)}_{\text{shift}}(\boldsymbol{z}_{\text{A}}, \boldsymbol{c}_{\text{br}}, \boldsymbol{\xi}^{(i)}_{\text{shift}})\Big). \end{aligned} \tag{21}$$

Following the main model, we apply a hyperbolic tangent function with learnt scale as the final activation of $\text{MLP}^{(i)}_{\text{scale}}$ for training stability [13]. This construction preserves differentiability in t and guarantees the consistency at the boundary conditions ($\tau = 0, 1$) where $\alpha(\tau) = 0$.

B Derivation of the Flow Loss

In our research, we utilise a bridge distribution described in Appendix A.2 as an auxiliary variational distribution [1, 40, 45] to constrain the upper bounds of bidirectional KL divergences between the one-step and two-step distributions of the NSF. By minimising these upper bounds, we guide the model towards satisfying the Chapman–Kolmogorov relation, as shown in Equation (4), thereby improving its consistency with the theoretical requirements of stochastic flows of diffeomorphisms combined with architecture design. Specifically, we define $\mathcal{L}_{\text{flow}}$ as the expectation of a weighted sum of two components:

$$\mathcal{L}_{\text{flow}}(\boldsymbol{\theta}, \boldsymbol{\xi}) = \mathop{\mathbb{E}}_{p(t_i, t_j, t_k)} \left[\lambda_{\text{1-to-2}} \mathcal{L}_{\text{flow, 1-to-2}}(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k) + \lambda_{\text{2-to-1}} \mathcal{L}_{\text{flow, 2-to-1}}(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k)\right]. \tag{22}$$

Here, $\lambda_{\text{1-to-2}}$ and $\lambda_{\text{2-to-1}}$ are weighting factors, which are set to 1 in the main text, balancing the contribution of each component to the overall flow loss. The two components, $\mathcal{L}_{\text{flow, 1-to-2}}$ and $\mathcal{L}_{\text{flow, 2-to-1}}$, correspond to the upper bounds of the one-step to two-step and two-step to one-step KL divergences, respectively.

In our experiment, which focuses on the autonomous case, the triplet (t_i, t_j, t_k) is sampled according to the following procedure:

- The initial time t_i is sampled from the data $\mathcal{D}$, representing the starting times in the dataset. (For non-autonomous SDEs, it is recommended to sample t_i uniformly to ensure the Chapman–Kolmogorov equation is satisfied across all time points.)
- The final time t_k is sampled from a mixture distribution $p(t_k) = \frac{1}{2}\mathcal{U}(t_i, t_i + H_{\text{train}}) + \frac{1}{2}\, p_{\text{data}}(t_k \mid t_i)$, where $\mathcal{U}(t_i, t_i + H_{\text{train}})$ denotes the uniform distribution over the interval $[t_i, t_i + H_{\text{train}}]$, and $p_{\text{data}}(t_k \mid t_i)$ is the conditional data distribution of end times corresponding to the sampled initial time t_i. Here, H_{train} represents the maximum one-shot interval used during training.
- The intermediate time t_j is uniformly sampled from the interval $[t_i, t_k]$, i.e., $t_j \sim \mathcal{U}(t_i, t_k)$.

The inclusion of $p_{\text{data}}(t_k \mid t_i)$ in the mixture distribution for sampling t_k is motivated by our observation that the model achieves higher accuracy in regions where data exists. By incorporating this data-driven component, we aim to enhance the efficiency of the learning process, leveraging the model's improved performance in data-rich areas.

In the following, we derive the upper bounds of the one-step to two-step and two-step to one-step KL divergences.

B.1 One-Step to Two-Step KL Divergence

We begin with the derivation of the upper bound of the KL divergence from a one-step computation to a marginalised two-step computation. To clarify the computation, subscripts $p^1_{\boldsymbol{\theta}}$ and $p^2_{\boldsymbol{\theta}}$ distinguish between the distributions for 1-step sampling from $\boldsymbol{x}_{t_i}$ to $\boldsymbol{x}_{t_k}$ and two-step sampling from $\boldsymbol{x}_{t_i}$ to $\boldsymbol{x}_{t_k}$ via $\boldsymbol{x}_{t_j}$, and the time arguments are omitted from the probability distributions for clarity. The key steps are outlined as follows:

$$D_{\mathrm{KL}}\left(p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}) \,\middle\|\, \int p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j}\right) \tag{23}$$

$$= \int p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}) \log \left[\frac{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})}{\int p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j}}\right] \mathrm{d}\boldsymbol{x}_{t_k} \tag{24}$$

$$= \int p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}) \log \left[\frac{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}\right] \mathrm{d}\boldsymbol{x}_{t_k} \tag{25}$$

$$\begin{aligned} = \iint p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\, q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k}) \Bigg[&\log \left[\frac{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\, q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}\right] \\ &- \log \left[\frac{q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}\right] \Bigg] \mathrm{d}\boldsymbol{x}_{t_j} \mathrm{d}\boldsymbol{x}_{t_k} \end{aligned} \tag{26}$$

$$\begin{aligned} = \mathop{\mathbb{E}}_{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}) q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})} \Bigg[&\log \left[\frac{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\, q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}\right] \\ &- D_{\mathrm{KL}}\big(q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k}) \,\big\|\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})\big) \Bigg] \end{aligned} \tag{27}$$

$$\leq \mathop{\mathbb{E}}_{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i}) q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})} \left[\log \left[\frac{p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\, q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}\right]\right] \tag{28}$$

$$=: \mathcal{L}_{\text{flow, 1-to-2}}(\boldsymbol{\theta}, \boldsymbol{\phi}; t_i, t_j, t_k). \tag{29}$$

https://doi.org/10.52202/085713-0571

The equation from the second line to the third line is obtained using Bayes' theorem and the Markov property:

$$\int p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j} = \frac{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})}. \tag{30}$$

B.2 Two-Step to One-Step KL Divergence

Similarly, we now explore the derivation of the upper bound of the KL divergence from a marginalised two-step computation to a one-step computation. We denote this as $\mathcal{L}_{\text{flow, 2-to-1}}$. The key steps are outlined as follows:

$$D_{\mathrm{KL}}\left(\int p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j} \;\middle\|\; p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})\right) \tag{31}$$

$$= \int \left[\int p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, \mathrm{d}\boldsymbol{x}_{t_j}\right] \log \left[\frac{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})\, p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})}\right] \mathrm{d}\boldsymbol{x}_{t_k} \tag{32}$$

$$= \iint p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j}) \log \left[\frac{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})}{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})\, p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})}\right] \mathrm{d}\boldsymbol{x}_{t_j} \mathrm{d}\boldsymbol{x}_{t_k} \tag{33}$$

$$= \mathop{\mathbb{E}}_{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}) p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})} \left[\log \left[\frac{p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i})\, p^2_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_j})}{b_{\boldsymbol{\xi}}(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})\, p^1_{\boldsymbol{\theta}}(\boldsymbol{x}_{t_k} \mid \boldsymbol{x}_{t_i})}\right]\right] \tag{34}$$

$$=: \mathcal{L}_{\text{flow, 2-to-1}}(\boldsymbol{\theta}, \boldsymbol{\xi}; t_i, t_j, t_k). \tag{35}$$

From the first line to the second line, we use the relationship in Equation (30). Although the expression inside the log in the second line appears to depend on $\boldsymbol{x}_{t_j}$, it does not actually depend on $\boldsymbol{x}_{t_j}$ due to this relationship. Therefore, the integrand change from the second line to the third line is justified.

By minimising flow loss consists of these upper bounds, we encourage the model to satisfy the flow property in both directions simultaneously.

C Detailed Description of the Learning Process

C.1 Neural Stochastic Flow Optimisation

The optimisation process of NSF is summarised in Algorithm 1 in the main text. Here, we provide additional implementation details.

1. **Batch and time triplet sampling**: We sample a batch $\{(\boldsymbol{x}_{t_i}, t_i)_b\}_{b=1}^B$ from the dataset. For each element, we sample a triplet of time points (t_i, t_j, t_k) where $t_i < t_j < t_k$, following the strategy described in Appendix B.
2. **Bridge model optimisation**: Before updating the main model parameters, we perform K inner optimisation steps on the bridge model parameters $\boldsymbol{\xi}$. This ensures that the auxiliary variational distribution $b_{\boldsymbol{\xi}}(\boldsymbol{x}_{t_j} | \boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_k})$ closely approximates the model's bridge distribution.
3. **Main model optimisation**: We then sample transition pairs $\{(\boldsymbol{x}_{t_i}, \boldsymbol{x}_{t_j}, t_i, t_j)_b\}_{b=1}^B$ from the dataset and compute the negative log-likelihood. Combined with the flow loss, this forms the total loss used to update the main model parameters $\boldsymbol{\theta}$.

This sequencing ensures that the main model is updated based on a bridge model that closely approximates the bridge distribution of the main model. Alternatively, the inner optimisation steps can be omitted and update the bridge model parameters simultaneously with the main model parameters when the main model parameters are updated.

C.2 Latent Neural Stochastic Flow Optimisation

The optimisation process of Latent NSF is summarised in Algorithm 1 in the main text. Here, we provide additional implementation details.

1. **Observation encoding**: We sample batches of observation sequences $\{\boldsymbol{o}_{t_{0:T},b}\}_{b=1}^{B}$ and encode them to latent states using the encoder network $q_{\boldsymbol{\phi}}(\boldsymbol{x}_{t_{0:T},b}|\boldsymbol{o}_{t_{0:T},b})$ as defined in Eq. (15).
2. **Time triplet sampling**: Similar to NSF, we sample time triplets (t_i, t_j, t_k) for each sequence, with $t_i < t_j < t_k$.
3. **Bridge model optimisation**: Before updating the main model parameters, we perform K inner optimisation steps on the bridge model parameters $\boldsymbol{\xi}$.
4. **Main model optimisation**: We then compute the total loss using Eq. (18) in the main text and update the NSF parameters $\boldsymbol{\theta}$, encoder parameters $\boldsymbol{\phi}$, and decoder parameters $\boldsymbol{\psi}$.

The specific values of hyperparameters β, β_{skip}, and λ in the total loss (Eq. (18) in the main text) vary depending on the dataset and task, and are detailed in Appendix E. As well as NSF, the inner optimisation steps can be omitted and update the auxiliary model parameters simultaneously with the main model parameters when the main model parameters are updated.

D Sampling and Density Evaluation

To generate a sample at time t_j from a known state $\boldsymbol{x}_{t_i}$ at time t_i, we compute $\boldsymbol{\mu}(\boldsymbol{x}_{t_i}, t_i, \Delta t)$ and $\boldsymbol{\sigma}(\boldsymbol{x}_{t_i}, t_i, \Delta t)$ using the MLPs, draw $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and compute $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$. Then, apply the flow transformation: $\boldsymbol{x}_{t_j} = \boldsymbol{f}_L \circ \boldsymbol{f}_{L-1} \circ \cdots \circ \boldsymbol{f}_1(\boldsymbol{z}; \Delta t, t_i)$. This enables single-step prediction without numerical integration, following the standard approach used in normalising flows.

Using the change of variables formula, the log-density can be computed as:

$$\log p_\theta\big(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, \Delta t\big) = \log p(\boldsymbol{\varepsilon}) - \sum_{i=1}^{L} \log \left| \det \frac{\partial f_i}{\partial f_{i-1}} \right|, \tag{36}$$

where $p(\boldsymbol{\varepsilon})$ is the density of the standard normal distribution. When using the affine coupling formulation, this simplifies to:

$$\log p_\theta\big(\boldsymbol{x}_{t_j} \mid \boldsymbol{x}_{t_i}; t_i, \Delta t\big) = -\frac{1}{2}\|\boldsymbol{\varepsilon}\|^2 - \frac{d}{2}\log 2\pi - \sum_{i=1}^{L} \sum_{j \in B_i} \Delta t \cdot \text{MLP}^{(i)}_{\text{scale}}(\boldsymbol{z}_{\text{A}}^{(i-1)}, t_i, \Delta t)_j, \tag{37}$$

where B_i is the set of indices corresponding to the second partition in the i-th coupling layer, and $\boldsymbol{z}^{(i-1)}$ is the output of the $(i-1)$-th layer. This density evaluation approach mirrors the standard technique in normalising flows, with appropriate conditioning on the time parameters.

E Experimental Details

This section provides detailed information about the experimental setups, hyperparameters, architectures, and computational analyses for the experiments presented in Section 6 in the main text. Implementations are available at `https://github.com/nkiyohara/jax_nsf`.

E.1 General Setup

Unless otherwise specified, experiments were conducted with the following setup:

- Frameworks:
  - Our (Latent) NSF models: JAX [5] with Equinox library [24]
  - PyTorch baselines: torchsde [33]
  - JAX baselines: Diffrax [23]
- Hardware: NVIDIA RTX 3090 GPU
- Optimiser: AdamW [27, 37]
- Hyperparameters: Specified below for each experiment

https://doi.org/10.52202/085713-0571

E.2 Stochastic Lorenz Attractor

E.2.1 Data Generation

Following Li et al. [33], the data generation process is as follows:

- SDE:
$$\begin{aligned} \mathrm{d}x &= \sigma(y - x)\,\mathrm{d}t + \alpha_x\,\mathrm{d}W_1 \\ \mathrm{d}y &= (x(\rho - z) - y)\,\mathrm{d}t + \alpha_y\,\mathrm{d}W_2 \\ \mathrm{d}z &= (xy - \beta z)\,\mathrm{d}t + \alpha_z\,\mathrm{d}W_3 \end{aligned}$$
- Parameters: $\sigma = 10, \rho = 28, \beta = 8/3, \alpha = (0.15, 0.15, 0.15)$
- Initial states (x_0, y_0, z_0): Sampled from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- Trajectories: 1,024 for training and 1,024 for testing
- Time steps: $t \in [0, 1]$ with time step size $\Delta t = 0.025$ (41 time steps per trajectory)

E.2.2 Configurations for Neural Stochastic Flows

- Architecture:
  - State dimension `d_state = 3`; conditioning dimension `d_cond = 4` (state + time)
  - Gaussian parameter network: `Input(d_cond)` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(2*d_state)`; splits into (`mean`, `std`) with std via `Softplus()`
  - Scale-shift networks (in conditional flow): `Input(d_state/2 + d_cond)` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(d_state)`; splits into (`scale`, `shift` in Eq. (7) in the main text) (In this case, since `d_state` is odd, we split three dimensions of `d_state` into one and two dimensions)
  - Conditional flow (affine coupling; Eq. (5), (6)): 4 layers with alternating masking
- Parameters:
  - NSF: 25,130
  - Bridge model: 26,410
- Training:
  - Data conversion: Time series data is converted into pairs of states (x_{t_i}, x_{t_j}) where $t_j > t_i$, to predict $p(x_{t_j} \mid x_{t_i})$
  - Epochs: 1000
  - Batch size: 256
  - Optimiser: AdamW [27, 37]
  - Learning rate: 0.001
  - Weight decay: 10^{-5}
  - Loss function: Combined negative log-likelihood and flow loss (Eq. (11))
    - $\lambda = 0.4$ (0.2 for data component, 0.2 for sampled component)
    - $\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1.0$
  - Time triplets (t_i, t_j, t_k) for $\mathcal{L}_{\text{flow}}$: Sampled as described in Appendix B with $H_{\text{train}} = 1$
  - Auxiliary updates: bridge model $b_{\boldsymbol{\xi}}$ trained concurrently with $K = 5$ inner optimisation steps per main model update

E.2.3 Configurations for Baselines

For our baseline comparisons, we utilised the official implementations of latent SDE [33], SDE matching [3], SDE-GAN [25], and Stable Neural SDE series [38] with minimal modifications. We maintained the original model architectures, optimiser configurations, and hyperparameter settings as provided in their respective codebases, replacing only the input data to match our experimental setting. For SDE-GAN, we found that it exhibited significant training instability when applied to the Stochastic Lorenz Attractor dataset, preventing convergence despite multiple training attempts with varying hyperparameter configurations. Therefore, we excluded SDE-GAN results from our comparison in the paper.

Latent SDE. The latent SDE model [33] has the following configuration:

- Architecture:
  - Observation dimension `d_obs = 3`; latent dimension `d_latent = 4`; hidden size `d_hidden = 128`; sequence length `T = 41`
  - Encoder: `Input((T, d_obs))` $\xrightarrow[\text{reverse in time}]{}$ `GRU(128)` $\rightarrow$ `Linear(d_context)` $\xrightarrow[\text{restore time order}]{}$ `Output((T, d_context))`
  - Posterior drift network: `Input(d_latent + d_context)` $\rightarrow 2\times$[`Linear(128)` $\rightarrow$ `Softplus()`] $\rightarrow$ `Linear(d_latent)`
  - Prior drift network: `Input(d_latent)` $\rightarrow 2\times$[`Linear(128)` $\rightarrow$ `Softplus()`] $\rightarrow$ `Linear(d_latent)`
  - Diffusion networks: per-dimension `Input(1)` $\rightarrow$ `Linear(128)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(1)` $\rightarrow$ `Sigmoid()`
  - Decoder: `Input(d_latent)` $\rightarrow$ `Linear(d_obs)`
- Parameters:
  - Encoder: 59,328
  - Posterior initial state network: 520
  - Posterior drift network: 25,860
  - Prior drift network: 17,668
  - Diffusion networks: 1,540
  - Decoder: 15
- Training:
  - Iterations: 5000
  - Optimiser: Adam
  - Initial learning rate: 0.01
  - Learning rate decay: Exponential (gamma=0.997)
  - KL divergence annealing: Linear annealing from 0 to 1 over first 1000 iterations
  - Numerical integration: Euler–Maruyama method with step size $\Delta t = 0.01$

SDE Matching. The SDE matching model [3] has the following configuration:

- Architecture:
  - Observation dimension `d_obs = 3`; latent dimension `d_latent = 4`; hidden size `d_hidden = 128`
  - Encoder: `Input((T, d_obs))` $\rightarrow$ `GRU(d_hidden)` $\rightarrow$ `Output((T+1, d_hidden))` (concatenate initial hidden with per-step outputs)
  - Prior initial distribution: learnable diagonal Gaussian; parameters `m`, `log s` in `d_latent` dimensions each
  - Prior drift network: `Input(d_latent)` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`
  - Diffusion networks: per-dimension `Input(1)` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(1)` $\rightarrow$ `Sigmoid()`
  - Posterior affine head: `Input(d_hidden + 1)` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(2*d_latent)`; splits into $(\mu_t, \log \sigma_t)$ with $\sigma_t = \exp(\log \sigma_t)$
  - Decoder: `Input(d_latent)` $\rightarrow$ `Linear(d_obs)`; Gaussian likelihood with fixed std $\sigma = 0.01$
- Parameters:
  - Prior init: 8
  - Prior drift network: 17,668
  - Diffusion networks: 1,540

 https://doi.org/10.52202/085713-0571

  - Decoder: 15
  - Encoder: 99,072
  - Posterior affine head: 66,560
- Training:
  - Iterations: 10,000
  - Batch size: 1,024
  - Optimiser: Adam
  - Learning rate: 0.001

Stable Neural SDE Series. The Stable Neural SDE series includes Neural LSDE, Neural GSDE, and Neural LNSDE.

- Architecture (common to all variants unless specified):
  - Encoder: `Input(3 + 1)` $\to$ `Linear(64)`
  - Embedding network: `Input(64)` $\to$ `Linear(64)`
  - Drift network: `Input(64)` $\to 2\times$[`Linear(64)` $\to$ `LipSwish()`] $\to$ `Linear(64)`
  - Diffusion network: `Input(64)` $\to 2\times$[`Linear(64)` $\to$ `LipSwish()`] $\to$ `Linear(64)`
  - Decoder: `Input(64)` $\to$ `Linear(3)`
- Parameters:
  - Encoder: 256
  - Embedding network: 4,160
  - Drift network: 12,480
  - Diffusion network: 12,480
  - Decoder: 192
- Training:
  - Epochs: 100
  - Batch size: 16
  - Optimiser: Adam
  - Learning rate: 0.001
  - Numerical integration: Euler–Maruyama scheme with step-size $\Delta t = 0.01$

E.2.4 Evaluation and Metrics

KL Divergence. For computing the KL divergence, we employed non-parametric Kernel Density Estimation (KDE) with Gaussian kernels. The bandwidth parameter was determined through 3-fold cross-validation by optimising the log-likelihood on unseen data. To ensure statistical robustness, we computed the KL divergence and test log-likelihood across ten independent folds, excluding values beyond the 5th and 95th percentiles to mitigate outlier influence. The final reported metrics consist of the mean and standard deviation of these filtered values.

FLOPs. We calculated FLOPs on a per-sample basis using different methods depending on the framework. For JAX-based models like Neural Stochastic Flows, we analysed the JIT-compiled sampling function using `jax.jit().lower().compile().cost_analysis()['flops']`. For PyTorch-based models such as latent SDE and Stable Neural SDEs, we utilised `torch.utils.flop_counter.FlopCounterMode` to count FLOPs during sample generation, then divided by batch size. Our comparison table shows the FLOPs value for one sample.

Runtime. For all methods, we measured the wall-clock runtime for computing vectorised batches of 100 samples.

E.2.5 Detailed Results

Table 6 presents FLOPs comparisons across different prediction horizons. NSF requires constant FLOPs for single-step predictions regardless of time interval, while baseline methods scale linearly with the number of integration steps.

Table 7 shows wall-clock runtime measurements for generating 100 samples. PyTorch implementations exhibit higher runtime due to dynamic graph construction, while JAX implementations benefit from JIT compilation. NSF achieves consistent sub-millisecond runtime across all time horizons.

E.3 CMU Motion Capture

E.3.1 Data Preparation

Data was prepared as follows:

- Dataset: Preprocessed CMU Motion Capture (walking) from Yildiz et al. [54]
- Features: 50-dimensional joint angle vectors
- Sequence length: 300 time steps
- Split: 16 training, 3 validation, and 4 test sequences
- Setups:
  - Setup 1 [33]: Train on full 300 steps, test by predicting steps 4–300 given steps 1–3
  - Setup 2 [2]: Train on steps 1–200, test by predicting steps 101–300 given steps 1–100

E.3.2 Latent SDE Reproduction

As noted in the footnote 1 in the main text, whilst Li et al. [33] report 4.03 ± 0.20 for Setup 1, our re-implementation and other reproductions observe higher MSE values. These results are consistent with those reported in public discussions `https://github.com/google-research/torchsde/issues/112`. Our implementation is available at `https://github.com/nkiyohara/latent-sde-mocap`, which is consistent with the results reported in the public discussion.

E.3.3 SDE Matching Reproduction

As noted in the footnote 2 in the main text, at the time of writing, the official implementation of SDE matching [3] for the CMU Motion Capture dataset has not yet been publicly released. To ensure consistent comparison across Setup 1 and Setup 2, we re-implemented SDE matching to the best of our ability based on the description in their paper. While Bartosh et al. [3] report 4.50 ± 0.32 for Setup 2, our re-implementation achieves 5.20 ± 0.43. This discrepancy may stem from implementation details not specified in the original paper, such as architectural choices, hyperparameter settings, or training procedures. We have made our best efforts obtain the smaller MSE by adjusting the hyperparameters, architecture, and training procedure. Our implementation is available at `https://github.com/nkiyohara/sde-matching-mocap`.

E.3.4 Configurations for Latent NSF

Latent NSF — Setup 1 (within-horizon forecasting).

- Architecture:
  - Latent dimension `d_latent = 6`; conditioning dimension `d_cond = 7`
  - Encoder: `Input(150)` $\rightarrow$ $2\times$`[Linear(30)` $\rightarrow$ `Softplus()]` $\rightarrow$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
  - Emission $p_{\psi}(\boldsymbol{o}_{t_i} \mid \boldsymbol{x}_{t_i})$: `Input(d_latent)` $\rightarrow$ $2\times$`[Linear(30)` $\rightarrow$ `Softplus()]` $\rightarrow$ `Linear(50)`; Gaussian likelihood with fixed std
  - NSF transition model (latent prior):
    - Gaussian parameter network: `Input(d_cond)` $\rightarrow$ $2\times$`[Linear(30)` $\rightarrow$ `SiLU()]` $\rightarrow$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`

https://doi.org/10.52202/085713-0571

Table 5: FLOPs comparison (K) for Stochastic Lorenz Attractor experiments, showing computational costs per single sample prediction. EM refers to Euler–Maruyama method, and SRK refers to Stochastic Runge–Kutta method.

Method	$t = 0.25$	$t = 0.5$	$t = 0.75$	$t = 1.0$
Latent SDE [33]				
EM, $\Delta t = 0.003$	3,133	6,193	9,253	12,349
EM, $\Delta t = 0.01$	959	1,917	2,839	3,760
EM, $\Delta t = 0.03$	369	664	995	1,290
Milstein, $\Delta t = 0.01$	1,010	2,021	2,994	3,967
SRK, $\Delta t = 0.01$	9,253	18,838	28,054	37,270
SDE matching [3]				
EM, $\Delta t = 0.0001$	184,394	368,787	553,034	737,354
EM, $\Delta t = 0.001$	18,506	37,011	55,443	73,875
EM, $\Delta t = 0.01$	1,917	3,834	5,677	7,520
Stable Neural SDEs [38]				
Neural LSDE (EM, $\Delta t = 0.01$)	1,708	3,416	5,057	6,699
Neural LSDE (Milstein, $\Delta t = 0.01$)	1,708	3,416	5,057	6,699
Neural LSDE (SRK, $\Delta t = 0.01$)	16,483	33,555	49,971	66,387
Neural GSDE (EM, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural GSDE (Milstein, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural GSDE (SRK, $\Delta t = 0.01$)	18,571	37,807	56,303	74,799
Neural LNSDE (EM, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural LNSDE (Milstein, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural LNSDE (SRK, $\Delta t = 0.01$)	18,571	37,807	56,303	74,799
NSF (ours)				
$H_{\text{train}} = 1.0, H_{\text{pred}} = 0.25$	53	105	156	208
$H_{\text{train}} = 1.0, H_{\text{pred}} = 0.5$	53	53	105	105
$H_{\text{train}} = 1.0, H_{\text{pred}} = 1.0$	53	53	53	53

Table 6: FLOPs comparison (K) for Stochastic Lorenz Attractor experiments, showing computational costs per single sample prediction. EM refers to Euler–Maruyama method, and SRK refers to Stochastic Runge–Kutta method.

Method	$t = 0.25$	$t = 0.5$	$t = 0.75$	$t = 1.0$
Latent SDE (EM, $\Delta t = 0.01$)	959	1,917	2,839	3,760
Latent SDE (Milstein, $\Delta t = 0.01$)	1,010	2,021	2,994	3,967
Latent SDE (SRK, $\Delta t = 0.01$)	9,253	18,838	28,054	37,270
Neural LSDE (EM, $\Delta t = 0.01$)	1,708	3,416	5,057	6,699
Neural LSDE (Milstein, $\Delta t = 0.01$)	1,708	3,416	5,057	6,699
Neural LSDE (SRK, $\Delta t = 0.01$)	16,483	33,555	49,971	66,387
Neural GSDE (EM, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural GSDE (Milstein, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural GSDE (SRK, $\Delta t = 0.01$)	18,571	37,807	56,303	74,799
Neural LNSDE (EM, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural LNSDE (Milstein, $\Delta t = 0.01$)	1,925	3,848	5,698	7,548
Neural LNSDE (SRK, $\Delta t = 0.01$)	18,571	37,807	56,303	74,799
NSF ($H_{\text{train}} = 0.25, H_{\text{pred}} = 0.25$)	53	105	156	208
NSF ($H_{\text{train}} = 0.5, H_{\text{pred}} = 0.5$)	53	53	105	105
NSF ($H_{\text{train}} = 1.0, H_{\text{pred}} = 1.0$)	53	53	53	53

    - Scale-shift network (in conditional flow): 4 layers with alternating masking, each layer: `Input(d_latent/2 + d_cond)` $\to$ $2\times$`[Linear(30)` $\to$ `SiLU()]` $\to$ `Linear(d_latent)`; splits into (`scale`, `shift` in Eq. (7) in the main text)
  - Bridge model:
    - Gaussian parameter network: `Input(d_cond + 1)` $\to$ $2\times$`[Linear(30)` $\to$ `SiLU()]` $\to$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
    - Scale-shift network (in conditional flow): 4 layers with alternating masking, each layer: `Input(d_latent + d_cond + 1)` $\to$ $2\times$`[Linear(30)` $\to$ `SiLU()]` $\to$ `Linear(d_latent)`; splits into (`scale`, `shift` in Eq. (20))
- Parameters:
  - Stochastic flow: 8,454
  - Bridge model: 9,504
  - Decoder: 2,690
  - Posterior: 5,832
- Training:
  - Objective: maximise Eq. (18)
  - Steps: 100,000
  - Batch size: 128
  - Optimiser: AdamW
  - Learning rate: 0.001
  - Hyperparameters: $\beta = 0.1$, $\lambda = 1.0$, $\beta_{\text{skip}} = 0.1$
  - Skip-ahead KL: prediction horizons up to 3 time steps; 10 samples per skip

Latent NSF — Setup 2 (extrapolation).

- Architecture:
  - Latent dimension `d_latent = 6`, conditioning dimension `d_cond = 7`
  - Encoder: `Input((T, 50))` $\to$ `GRU(32)` $\to$ $2\times$`[Linear(32)` $\to$ `Softplus()]` $\to$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
  - Emission $p_{\boldsymbol{\psi}}(\boldsymbol{o}_{t_i} \mid \boldsymbol{x}_{t_i})$: `Input(d_latent)` $\to$ $2\times$`[Linear(30)` $\to$ `Softplus()]` $\to$ `Linear(50)`; Gaussian likelihood with trainable std
  - NSF transition model (latent prior):
    - Gaussian parameter network: `Input(d_cond)` $\to$ $2\times$`[Linear(64)` $\to$ `SiLU()]` $\to$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
    - Scale-shift network (in conditional flow): 4 layers with alternating masking, each layer: `Input(d_latent/2 + d_cond)` $\to$ $2\times$`[Linear(64)` $\to$ `SiLU()]` $\to$ `Linear(d_latent)`; splits into (`scale`, `shift`)
  - Bridge model:
    - Gaussian parameter network: `Input(d_cond + 1)` $\to$ $2\times$`[Linear(32)` $\to$ `SiLU()]` $\to$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
    - Scale-shift network (in conditional flow): 4 layers with alternating masking, each layer: `Input(d_latent/2 + d_cond + 1)` $\to$ $2\times$`[Linear(32)` $\to$ `SiLU()]` $\to$ `Linear(d_latent)`; splits into (`scale`, `shift`)
- Parameters:
  - Stochastic flow: 28,820
  - Bridge model: 10,452
  - Decoder: 4,240
  - Posterior: 10,604
- Training:
  - Objective: maximise Eq. (18)

https://doi.org/10.52202/085713-0571

- Steps: 100,000
- Batch size: 64
- Optimiser: AdamW
- Learning rate: 0.001
- Hyperparameters: $\beta = 0.3$, $\lambda = 1.0$, $\beta_{\text{skip}} = 0.3$
- Warm-up: β and flow loss weight are linearly increased over the first 2,000 steps
- Time horizon for flow loss: $H_{\text{train}} = 20$
- Auxiliary updates: $K = 3$ inner steps per training iteration for the bridge model
- Skip-ahead KL: prediction horizons up to $\tau = 10$ time steps; 10 samples per skip

E.3.5 Configurations for Baselines

Latent SDE — Setup 1 (within-horizon forecasting).

- Architecture:
  - Observation dimension `d_obs = 50`; latent dimension `d_latent = 6`; context dimension `d_context = 3`
  - Encoder: `Input(150)` $\rightarrow 2\times$[`Linear(30)` $\rightarrow$ `Softplus()`] $\rightarrow$ `Linear(15)`; splits into (`mean`, `std`, `context`) with `std` via `exp()`, with 6, 6, and 3 dimensions respectively
  - Posterior drift network: `Input(d_latent + 1 + d_context)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`
  - Prior drift network: `Input(d_latent + 1)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`
  - Diffusion networks: per-latent `Input(2)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(1)` $\rightarrow$ `Sigmoid()`
  - Decoder: `Input(d_latent)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(2*d_obs)`
- Parameters:
  - Encoder: 5,925
  - Decoder: 4,240
  - Posterior drift network: 516
  - Prior drift network: 426
  - Diffusion networks: 726
- Training:
  - Iterations: 5,000
  - Optimiser: Adam
  - Learning rate: 0.01 with exponential decay $\gamma = 0.999$ at every step
  - KL annealing: linear over the first 400 iterations
  - Numerical integration: Euler–Maruyama with step size $\Delta t = 0.02$

Latent SDE — Setup 2 (extrapolation).

- Architecture:
  - Observation dimension `d_obs = 50`; latent dimension `d_latent = 10`; context dimension `d_context = 3`
  - Encoder: `Input((T, 50))` $\rightarrow$ `ODE-GRU(30)`; linear heads: `Input(30)` $\rightarrow$ `Linear(3)` and `Input(30)` $\rightarrow$ `Linear(20)`; splits into (`mean`, `std`) with `std` via `exp()`
  - Posterior drift network: autonomous `Input(d_latent + d_context)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`
  - Prior drift network: `Input(d_latent)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`

  - Diffusion networks: per-latent `Input(1)` $\to$ `Linear(30)` $\to$ `Softplus()` $\to$ `Linear(1)` $\to$ `Sigmoid()`
  - Decoder: `Input(d_latent)` $\to$ `Linear(30)` $\to$ `Softplus()` $\to$ `Linear(30)` $\to$ `Softplus()` $\to$ `Linear(2*d_obs)`
- Parameters:
  - Encoder: 12,027
  - Decoder: 4,360
  - Posterior drift network: 730
  - Prior drift network: 640
  - Diffusion networks: 910
- Training:
  - Iterations: $5{,}000$
  - Optimiser: Adam
  - Learning rate: 0.01 with exponential decay $\gamma = 0.9$ every 500 iterations
  - KL annealing: Linear over the first 500 iterations
  - Numerical integration: Euler–Maruyama with step size $\Delta t = 0.05$

SDE Matching – Setup 1 (within-horizon forecasting).

- Architecture:
  - Observation dimension `d_obs = 50`; latent dimension `d_latent = 6`; posterior hidden size `d_hidden = 100`
  - Encoder: `Input((T, d_obs))` $\xrightarrow[\text{reverse in time}]{}$ `GRU(d_hidden)` $\xrightarrow[\text{restore time order}]{}$ `Output((T+1, d_hidden))` (concatenate initial hidden with per-step outputs)
  - Posterior affine head: `Input(d_hidden + 1)` $\to$ `Linear(d_hidden)` $\to$ `SiLU()` $\to$ `Linear(d_hidden)` $\to$ `SiLU()` $\to$ `Linear(2*d_latent)`; splits into (μ_t, σ_t) with $\sigma_t = \texttt{Softplus}(\cdot)$; time-weighted aggregation over encoder outputs enabled
  - Prior initial distribution: learnable diagonal Gaussian; parameters `m`, `log_s` in $\mathbb{R}^{d\_latent}$
  - Prior drift network (autonomous): `Input(d_latent)` $\to$ `Linear(30)` $\to$ `Softplus()` $\to$ `Linear(d_latent)`
  - Diffusion networks (diagonal, per-latent): `Input(1)` $\to$ `Linear(30)` $\to$ `Softplus()` $\to$ `Linear(1)` $\to$ `Sigmoid()`
  - Decoder (observation model): `Input(d_latent)` $\to$ $2\times$[`Linear(30)` $\to$ `Softplus()`] $\to$ `Linear(d_obs)`; Gaussian likelihood with fixed std (σ_0=1.0, floor 10^{-3})
- Parameters:
  - Prior SDE: 942
  - Prior Observation: 2,740
  - Posterior Encoder: 45,400
  - Posterior Affine: 21,513
- Training:
  - Iterations: $2{,}000$
  - Optimiser: AdamW
  - Learning rate: 0.01 with exponential decay $\gamma = 0.999$ per step
  - KL weight: 1.0
  - Training sequence length: sampled from $\mathcal{U}\{3, 100\}$

SDE Matching – Setup 2 (extrapolation).

- Architecture:
  - Observation dimension `d_obs = 50`; latent dimension `d_latent = 10`; posterior hidden size `d_hidden = 1000`; prior SDE: autonomous

https://doi.org/10.52202/085713-0571

  - Encoder: `Input((T, d_obs))` $\xrightarrow[\text{reverse in time}]{}$ `GRU(d_hidden)` $\xrightarrow[\text{restore time order}]{}$ `Output((T+1, d_hidden))`
  - Posterior affine head: `Input(d_hidden)` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(d_hidden)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(2*d_latent)`; splits into (μ_t, σ_t) with $\sigma_t = \texttt{Softplus}(\cdot)$; time-weighted encoder aggregation enabled; no explicit time input
  - Prior initial distribution: learnable diagonal Gaussian in $\mathbb{R}^{d\_latent}$
  - Prior drift network (autonomous): `Input(d_latent)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(d_latent)`
  - Diffusion networks (diagonal, per-latent): `Input(1)` $\rightarrow$ `Linear(30)` $\rightarrow$ `Softplus()` $\rightarrow$ `Linear(1)` $\rightarrow$ `Sigmoid()`
  - Decoder (observation model): `Input(d_latent)` $\rightarrow$ 2$\times$[`Linear(30)` $\rightarrow$ `Softplus()`] $\rightarrow$ `Linear(d_obs)`; Gaussian likelihood with fixed std (σ_0=1.0, floor 10^{-3})
- Parameters:
  - Prior SDE: 1,550
  - Prior Observation: 2,860
  - Posterior Encoder: 3,154,000
  - Posterior Affine: 2,022,021
- Training:
  - Iterations: 100,000
  - Optimiser: AdamW
  - Learning rate: 0.0001
  - KL weight: 1.0
  - Sequence lengths: train=200, test=300;
  - Evaluation: Euler–Maruyama integration with fixed step $\Delta t = 0.01$

E.3.6 Evaluation and Metrics

We report the mean squared error (MSE) between the predicted mean trajectory and the ground truth joint angles over the forecast horizon (steps 4–300 for Setup 1, 101–300 for Setup 2). Results in Table 3 are averaged over 10 runs with different random seeds, reporting mean $\pm$ 95% t-confidence intervals where available from original papers or our runs.

E.3.7 Detailed Results

Table 8 provides FLOPs and runtime comparisons. Despite comparable computational complexity (FLOPs), Latent NSF achieves significantly faster runtime due to its ability to predict states at arbitrary time points in parallel, unlike traditional SDE methods that require sequential time-stepping.

E.4 Stochastic Moving MNIST

E.4.1 Data Generation

Data generation for Stochastic Moving MNIST was as follows:

- Frame size: 64x64 pixels
- Sequence length: 25 steps
- Digits: Two MNIST digits, chosen randomly
- Initial conditions: Positions and velocities sampled uniformly at random
- Bouncing mechanism: Modified from Denton and Fergus [11] for perfect reflection off boundaries with small angular noise $\mathcal{N}(0, \sigma^2_{\text{bounce}})$, where $\sigma_{\text{bounce}} = 0.1$ rad
- Dataset size:
  - Training: 60,000 sequences (using original MNIST training set)
  - Test: 10,000 sequences (using MNIST test set)

Table 7: Runtime comparison (ms) for Stochastic Lorenz Attractor experiments. All measurements are mean times for generating a batch of 100 vectorised samples (10 trials). EM refers to Euler–Maruyama method, and SRK refers to Stochastic Runge–Kutta method.

Method	Framework	$t = 0.25$	$t = 0.5$	$t = 0.75$	$t = 1.0$
Latent SDE (EM, $\Delta t = 0.01$)	PyTorch	141.7	281.8	417.8	1,988.3
Latent SDE (EM, $\Delta t = 0.01$)	JAX	123.5	130.7	150.6	147.8
Latent SDE (Milstein, $\Delta t = 0.01$)	PyTorch	493.4	951.2	1,449.9	3,294.2
Latent SDE (Milstein, $\Delta t = 0.01$)	JAX	142.7	149.4	181.4	188.7
Latent SDE (SRK, $\Delta t = 0.01$)	PyTorch	1,084.5	2,189.0	3,283.9	5,827.9
Neural LSDE (EM, $\Delta t = 0.01$)	PyTorch	101.0	198.9	291.6	1,632.7
Neural LSDE (Milstein, $\Delta t = 0.01$)	PyTorch	158.4	250.9	396.1	1,767.9
Neural LSDE (SRK, $\Delta t = 0.01$)	PyTorch	886.5	1,799.9	2,681.0	4,807.4
Neural GSDE (EM, $\Delta t = 0.01$)	PyTorch	107.6	215.4	316.9	1,662.3
Neural GSDE (Milstein, $\Delta t = 0.01$)	PyTorch	137.7	274.9	431.9	1,852.9
Neural GSDE (SRK, $\Delta t = 0.01$)	PyTorch	964.2	1,962.1	2,925.3	5,155.3
Neural LNSDE (EM, $\Delta t = 0.01$)	PyTorch	104.7	209.7	310.9	1,644.0
Neural LNSDE (Milstein, $\Delta t = 0.01$)	PyTorch	130.8	290.7	381.9	1,779.1
Neural LNSDE (SRK, $\Delta t = 0.01$)	PyTorch	950.0	1,921.4	2,853.7	5,023.7
NSF ($H_{\text{pred}} = 0.25$)	JAX	0.221	0.303	0.341	0.373
NSF ($H_{\text{pred}} = 0.5$)	JAX	0.229	0.287	0.332	0.310
NSF ($H_{\text{pred}} = 1.0$)	JAX	0.255	0.286	0.294	0.295

Table 8: FLOPs, runtime, and MSE comparison for CMU Motion Capture experiments. FLOPs are measured per sample (latent SDE uses Euler–Maruyama integration), while runtime measurements represent the time required to generate a batch of 100 vectorised samples.

Method	Framework	FLOPs (M)		Runtime (ms)		MSE	
		Setup 1	Setup 2	Setup 1	Setup 2	Setup 1	Setup 2
Latent SDE ($\Delta t = 0.005$)	JAX	13.5	25.1	3,585.0	3,236.0	–	–
Latent SDE ($\Delta t = 0.005$)	PyTorch	13.5	25.1	10,254.9	17,039.7	13.0	10.2
Latent SDE ($\Delta t = 0.01$)	JAX	8.0	17.8	1,894.0	1,664.0	–	–
Latent SDE ($\Delta t = 0.01$)	PyTorch	8.0	17.8	5,338.7	8,594.0	12.9	10.1
Latent SDE ($\Delta t = 0.02$)	JAX	5.2	14.1	854.2	909.6	–	–
Latent SDE ($\Delta t = 0.02$)	PyTorch	5.2	14.1	2,663.2	4,455.8	13.0	10.1
Latent SDE ($\Delta t = 0.05$)	JAX	3.6	11.9	374.1	494.2	–	–
Latent SDE ($\Delta t = 0.05$)	PyTorch	3.6	11.9	1,201.2	2,028.2	12.8	10.2
Latent SDE ($\Delta t = 0.1$)	JAX	3.0	11.2	223.0	372.6	–	–
Latent SDE ($\Delta t = 0.1$)	PyTorch	3.0	11.2	595.1	1,155.9	12.9	9.7
Latent SDE ($\Delta t = 0.2$)	JAX	2.7	10.8	148.4	323.3	–	–
Latent SDE ($\Delta t = 0.2$)	PyTorch	2.7	10.8	279.6	764.6	12.9	10.6
Latent SDE ($\Delta t = 0.5$)	JAX	2.6	10.6	107.1	289.6	–	–
Latent SDE ($\Delta t = 0.5$)	PyTorch	2.6	10.6	136.3	584.4	12.8	31.3
Latent SDE ($\Delta t = 1$)	JAX	2.5	10.5	99.88	279.6	–	–
Latent SDE ($\Delta t = 1$)	PyTorch	2.5	10.5	95.4	522.9	13.3	281.3
Latent SDE ($\Delta t = 2$)	JAX	2.5	10.5	92.19	266.1	–	–
Latent SDE ($\Delta t = 2$)	PyTorch	2.5	10.5	77.9	492.7	19.9	–
Latent SDE ($\Delta t = 5$)	JAX	2.5	10.5	89.13	269.7	–	–
Latent SDE ($\Delta t = 5$)	PyTorch	2.5	10.5	64.4	474.6	362.1	–
Latent NSF (ours)	JAX	6.0	17.7	0.44	1.8	8.7	3.4

https://doi.org/10.52202/085713-0571

E.4.2 Configurations for Latent NSF

- Architecture:
  - Latent dimension `d_latent = 10`; per-frame embedding dimension `d_embed = 64`; scene dimension `d_scene = 64`
  - Per-frame encoder (CNN, $64\times64 \rightarrow 4\times4$): 4 down blocks: `Conv2d(3*3, stride=1, padding=1)` $\rightarrow$ `MaxPool2d(2,2)` $\rightarrow$ `GroupNorm(8)` $\rightarrow$ `SiLU()`; channels: $1\rightarrow64\rightarrow128\rightarrow256\rightarrow256$; spatial: $64\rightarrow32\rightarrow16\rightarrow8\rightarrow4$; head: `Flatten()` $\rightarrow$ `Linear(d_embed)`
  - Posterior (inference network): `Input(d_embed)` $\rightarrow$ `GRU(128)`; head: `Input(128)` $\rightarrow$ `Linear(128)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
  - Scene encoder: temporal median pooling over per-frame embeddings; `Input(d_embed)` $\rightarrow$ `Linear(128)` $\rightarrow$ `SiLU()` $\rightarrow$ `Linear(d_scene)`
  - Emission: inputs `[scene, x_t]` with size `d_scene + d_latent`; `MLP()` $\rightarrow$ `Linear(4*4*4*d_embed)`; reshape to `(4*d_embed, 4, 4)`; 4 up blocks: `Conv(3*3)` $\rightarrow$ `GroupNorm(8)` $\rightarrow$ upsample(*2) $\rightarrow$ `SiLU()`, channels: `4*d_embed` $\rightarrow$ `4*d_embed` $\rightarrow$ `2*d_embed` $\rightarrow$ `d_embed`; spatial: `4` $\rightarrow$ `8` $\rightarrow$ `16` $\rightarrow$ `32`; then `Conv(3*3)` $\rightarrow$ `SiLU()` $\rightarrow$ `Conv(3*3)` $\rightarrow$ `Sigmoid()`; output size `C=1, H=W=64`; Gaussian likelihood with uniform trainable std (shared across pixels)
  - NSF transition model (latent prior):
    - Gaussian base: `Input()` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
    - Affine coupling flow: 4 layers with alternating masking (autonomous SDE)
    - Per-layer conditioner: `Input(d_latent/2)` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(d_latent)`; splits into (`scale`, `shift`)
  - Bridge model:
    - Gaussian base: `Input()` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(2*d_latent)`; splits into (`mean`, `std`) with std via `Softplus()`
    - Affine coupling flow: 4 layers with alternating masking
    - Per-layer conditioner: `Input(d_latent)` $\rightarrow$ 2$\times$[`Linear(64)` $\rightarrow$ `SiLU()`] $\rightarrow$ `Linear(d_latent)`
- Parameters:
  - Encoder: 1,223,360
  - Decoder: 1,341,570
  - Posterior: 110,228
  - Scene encoder: 33,088
  - NSF prior: 33,740
  - Bridge model: 37,260
- Training:
  - Epochs: 1,000
  - Batch size: 64
  - Learning rate: 0.001
  - Latent NSF objective parameters: $\beta = 30.0$, $\lambda = 0.1$, $\beta_{\text{skip}} = 30.0$, $H_{\text{train}} = 2.5$
  - Warm-up: β, β_{skip}, and flow loss weight linearly increased over the first 20 epochs
  - Auxiliary updates: $K = 3$ inner steps per iteration for the bridge model
  - Skip-ahead KL: up to 25-step horizons; 25 samples per skip

E.4.3 Configurations for Baseline

Latent SDE We used an implementation based on Daems et al. [10]. The baseline latent SDE model characteristics are as follows:

- Architecture:

  - Latent dimension `d_latent = 10`; per-frame embedding dimension `d_embed = 64`; content dimension `d_contents = 64`
  - Prior drift network: `Input(d_latent)` → `Linear(200)` → `Tanh()` → `Linear(200)` → `Tanh()` → `Linear(d_latent)`
  - Diffusion networks (per-latent): `Input(1)` → `Linear(200)` → `Tanh()` → `Linear(200)` → `Tanh()` → `Linear(1)` → `Softplus()`
  - Posterior control network: inputs `[x, y, h(t)]`; `Input(2*d_latent + d_embed)` → `Linear(200)` → `Tanh()` → `Linear(200)` → `Tanh()` → `Linear(d_latent)`; last layer kernel initialised to zero
  - Encoder (2D conv, frames `64*64*1`): four down blocks: `Conv(3*3)` → `MaxPool(2)` → `GroupNorm(8)` → `SiLU()`; channels `1` → `d_embed` → `2*d_embed` → `4*d_embed` → `4*d_embed`; spatial `64` → `32` → `16` → `8` → `4`; head: `Flatten()` → `Linear(d_embed)` per frame
  - Content extractor: temporal median pooling over per-frame embeddings; `Input(d_embed)` → `Linear(d_embed)` → `SiLU()` → `Linear(d_contents)`; outputs `w`
  - Inference network for `x_0`: temporal `Conv(3)` → `SiLU()` → `Conv(3)`; then `MLP()` on concatenated features to output `2*d_latent` (mean and log-variance)
  - Decoder: inputs `[w, x_t]` with size `d_contents + d_latent`; `MLP()` → `Linear(4*4*4*d_embed)`; reshape to `(4, 4, 4*d_embed)`; `4` up blocks: `Conv(3*3)` → `GroupNorm(8)` → upsample(*2) → `SiLU()`, with channels `4*d_embed` → `4*d_embed` → `2*d_embed` → `d_embed`; spatial `4` → `8` → `16` → `32`; then `Conv(3*3)` → `SiLU()` → `Conv(3*3)` → `Sigmoid()`; output size `C=1, H=W=64`
- Parameters:
  - Encoder: 1,269,972
  - Scene encoder: 8,320
  - Decoder: 1,341,569
  - Prior drift network: 44,410
  - Posterior drift network: 59,210
  - Diffusion network: 408,010
- Training:
  - Epochs: 100
  - Batch size: 32
  - Optimiser: Adam
  - Learning rate: 0.0003
  - KL weight: 0.1
  - Numerical integration: Stratonovich–Milstein with step size equal to one third of the observation interval

E.4.4 Evaluation and Metrics

Evaluation was performed using Fréchet distances with embeddings from the pre-trained SRVP model [14] (using publicly available weights). The computed metrics are:

- Static FD: Based on time-averaged frame embeddings
- Dynamics FD: Based on sequence-level dynamics embeddings
- Frame-wise FD: Averaged per-frame embedding distances

Validation of these metrics is provided in Appendix G.

E.4.5 Detailed Results

Table 9 provides computational comparisons for the Stochastic Moving MNIST experiments. Beyond the quantitative metrics reported in the main text, we provide qualitative visualisations to demonstrate

 https://doi.org/10.52202/085713-0571

the reconstruction and prediction quality of our Latent NSF model compared to the latent SDE baseline.

Figure 6 shows a direct comparison of reconstruction quality between latent SDE and Latent NSF on input sequences. Both models are tasked with encoding and then reconstructing the same 25-frame input sequences. The figure demonstrates that Latent NSF achieves comparable reconstruction quality while maintaining computational efficiency.

For predictive performance, Figure 7 illustrates the models' ability to forecast future frames given the first 25 frames of a sequence. The ground truth trajectory (top row) shows the natural bouncing dynamics of the two MNIST digits. The latent SDE predictions (middle row) and Latent NSF recursive predictions (bottom row) both capture the general trajectory and bouncing behaviour, though with different levels of visual fidelity and temporal consistency.

Finally, Figure 8 provides a focused comparison at a specific prediction horizon (20 steps into the future) by showing multiple predicted samples from each model. This visualisation allows for assessment of both the accuracy of individual predictions and the diversity of the stochastic predictions across different model variants.

F Hyperparameter Sensitivity and Ablation Studies

To validate the sensitivity of the hyperparameters and effectiveness of the flow loss, we conduct an ablation study on the Stochastic Lorenz Attractor with missing data (Section F.1), and CMU Motion Capture dataset with extrapolation task (Setup 2) (Section F.2).

F.1 Stochastic Lorenz Attractor with Missing Data

We trained on Stochastic Lorenz Attractor data where only a random continuous segment of length 0.5 from the full trajectory $t \in [0, 1]$ was observed.

F.1.1 Experimental Setup

We modify the training protocol as follows:

- **Original data structure**: The complete dataset contains trajectories with 41 time points at $t \in \{0, 0.025, 0.05, \ldots, 0.975, 1.0\}$, sampled at regular intervals of $\Delta t = 0.025$.
- **Missing data protocol**: We only use random continuous segment of length 20 time steps from the training data.
- **Training pairs**: From the 20 time points, we construct all possible state transition pairs (x_{t_i}, x_{t_j}) where $t_i < t_j$. This yields $20 \times 19/2 = 190$ unique pairs per trajectory, compared to 820 pairs in the complete data setting. Crucially, no training pairs directly connect states across > 20 time steps.
- **Training conditions**: All other hyperparameters remain identical to Section E.2.2:
  - Epochs: 1000
  - Optimiser: AdamW with learning rate 0.001 and weight decay 10^{-5}
  - Batch size: 256
  - Inner steps for auxiliary model: $K = 5$ inner optimisation steps
  - Flow loss weights: $\lambda = 0.4$
  - Flow loss balancing: $\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1.0$
- **Evaluation**: We compute KL divergence between the model and true distributions at $t \in \{0.5, 1.0\}$ using kernel density estimation as described in Section E.2.4. Results are averaged over 5 random seeds.

Based on this baseline condition, we conducted experiments varying the inner steps K, flow loss weight λ, and directional flow loss components $\lambda_{\text{1-to-2}}$ and $\lambda_{\text{2-to-1}}$.

Table 9: FLOPs and runtime comparison for Stochastic Moving MNIST experiments, showing computational costs for encoding 25 input frames and predicting the subsequent 25 frames. Runtime measurements represent the time required to process a batch of 100 vectorised samples. SM: Stratonovich–Milstein solver.

Method	Framework	FLOPs (M)	Runtime (ms)
Latent SDE (SM)	JAX	20,264	750
Latent NSF (recursive)	JAX	20,276	338
Latent NSF (one-step)	JAX	20,277	358

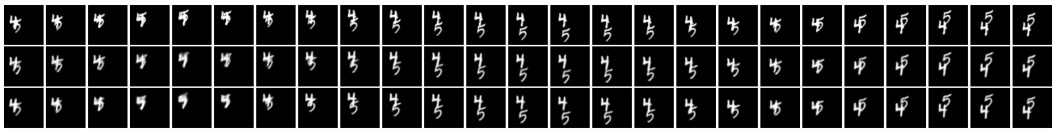


Figure 6: Reconstruction quality comparison on Stochastic Moving MNIST. Top row: original input frames, middle row: latent SDE reconstructions, bottom row: Latent NSF (recursive) reconstructions. Each column shows consecutive time steps (frames 1–25). Both models successfully capture the appearance and positioning of the bouncing digits.

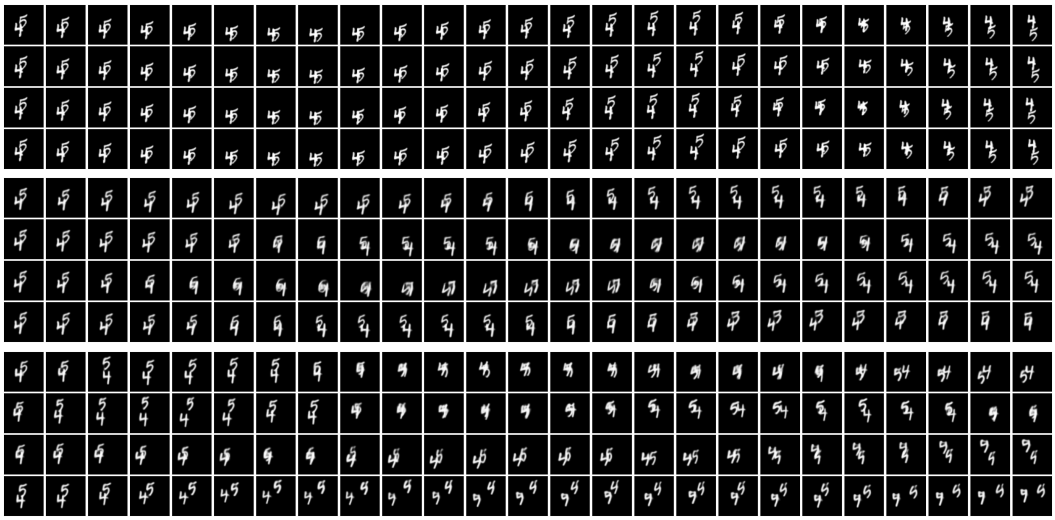
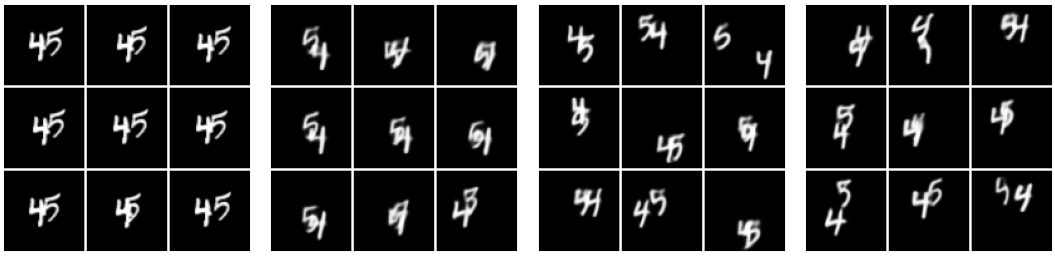


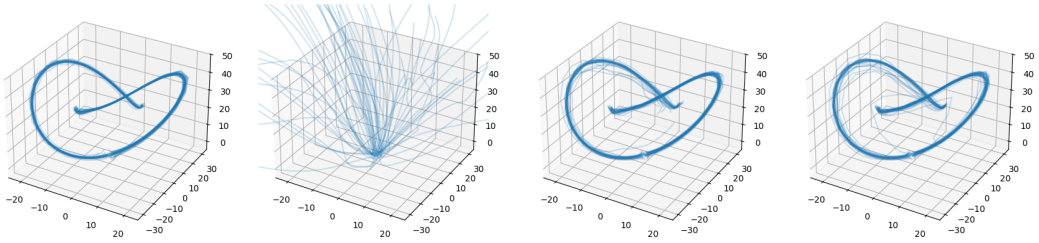
Figure 7: Future frame prediction comparison on Stochastic Moving MNIST. Top: ground truth continuation, middle: latent SDE predictions, bottom: Latent NSF predictions. Both models predict 25 future frames given the first 25 frames. The comparison shows how well each model captures stochastic bouncing dynamics while maintaining visual quality and temporal coherence.



(a) Ground truth (b) Latent SDE (c) Latent NSF (one-step) (d) Latent NSF (recursive)

Figure 8: Stochastic prediction samples 20 steps into the future on Stochastic Moving MNIST. Each panel shows multiple independent samples generated after observing the first 25 frames and predicting 20 steps ahead.

https://doi.org/10.52202/085713-0571



(a) Ground truth (b) $\lambda = 0.0$ (c) $\lambda = 0.2$ (d) $\lambda = 0.8$

Figure 9: Comparison of generated samples (64 samples per panel) on the stochastic Lorenz attractor. Columns show ground truth and model samples with different flow loss weights $\lambda \in \{0.0, 0.2, 0.8\}$.

Table 10: Flow loss ablation on missing Stochastic Lorenz. Reported values are mean $\pm$ standard deviation over seeds.

Flow loss weight (λ)	KL ($t = 0.5$)	KL ($t = 1.0$)
0.0 (no flow loss)	7.9 ± 0.2	22.9 ± 1.8
0.001	3.0 ± 0.2	4.5 ± 0.9
0.01	1.7 ± 0.1	2.1 ± 0.6
0.4 (default)	1.2 ± 0.1	1.2 ± 1.0
0.8	1.3 ± 0.2	1.3 ± 0.6

Table 11: Effect of auxiliary optimisation steps K on KL and training time (per epoch). $\lambda = 0.4$. Simultaneous: auxiliary and main models updated jointly without inner loops, as described in Appendix C.1.

K (auxiliary inner steps)	KL ($t = 0.5$)	KL ($t = 1.0$)	Epoch time (s)
0 (no auxiliary steps)	7.9 ± 0.2	22.9 ± 1.8	66
0 (simultaneous)	1.1 ± 0.1	1.8 ± 1.2	106
1	1.1 ± 0.1	1.7 ± 0.6	125
5 (default)	1.2 ± 0.1	1.4 ± 0.6	201
10	1.1 ± 0.2	1.2 ± 0.7	295

Table 12: Effect of directional flow loss components on KL. $\lambda = 0.4, K = 5$.

Flow loss direction	KL ($t = 0.5$)	KL ($t = 1.0$)
1-to-2 only $\lambda_{\text{1-to-2}} = 1, \lambda_{\text{2-to-1}} = 0$	1.1 ± 0.1	1.4 ± 0.6
2-to-1 only $\lambda_{\text{1-to-2}} = 0, \lambda_{\text{2-to-1}} = 1$	1.2 ± 0.1	1.4 ± 0.7
Bidirectional $\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1$	1.2 ± 0.1	1.4 ± 0.6

F.1.2 Results

Flow loss weight sensitivity is summarised in Table 10, and visualisation of 64 trajectories for the ground truth and generated samples is shown in Figure 9, which is obtained in the same way as Figure 3 in the main text with one-step prediction. The effect of auxiliary inner steps is shown in Table 11. Directional components are compared in Table 12. Overall, the sensitivity to the flow loss weight and the number of inner steps is low when the order of magnitude is around the baseline conditions.

F.2 CMU Motion Capture Dataset with Extrapolation Task (Setup 2)

F.2.1 Experimental Setup

We use the same experimental setup as Setup 2 in Section E.3.4, and conducted the parametric study varying the inner steps K, flow loss weight λ, and directional flow loss components $\lambda_{\text{1-to-2}}$ and $\lambda_{\text{2-to-1}}$ on the CMU Motion Capture dataset with extrapolation task (Setup 2).

F.2.2 Results

Flow loss weight (bidirectional, $K = 3$) is reported in Table 13. Directional variants for flow loss at $\lambda = 1.0$, $K = 3$ are summarised in Table 14. The number of inner steps for the auxiliary model (with $\lambda = 1.0$, bidirectional) are shown in Table 15. In short, best MSE occurs near $\lambda = 0.03$, direction 2-to-1 is slightly better than 1-to-2 but bidirectional is comparable, and small K (e.g., 1) also works while $K = 0$ fails, which means that, in this case, the sensitivity to the flow loss weight and the number of inner steps is low when the order of magnitude is around the baseline conditions.

F.3 Practical Guidance on Hyperparameters

We provide practical guidance on hyperparameters specifically for the NSF or Latent NSF.

F.3.1 Time Horizon for Training H_{train} and Inference H_{pred}

The time horizon for training should be set to the maximum one-shot interval used during training. In our experiments, we found that there is no significant difference in performance dependent on the H_{pred} value. However we have not explored the effect of the H_{train} value. Since, in practice, large H_{train} requires more expressive model capacity, we recommend balancing trade-off between model capacity and prediction runtime using H_{train}, and using $H_{\text{pred}} = H_{\text{train}}$ in most cases, depending on the data complexity and prediction runtime requirements.

F.3.2 Flow Loss Weight

For any experiment we have conducted, we found that sub-order of magnitude λ values are sufficient to achieve good performance, assuming the data is normalised.

F.3.3 Auxiliary Inner Steps or Simultaneous Updates

The auxiliary inner steps K controls the number of inner steps for the auxiliary model. We recommend using simultaneous updates first, as described in Appendix C.2, and if there is training instability, we recommend using $K > 0$ and increasing the value of K until the flow loss is stable.

F.3.4 Directional Flow Loss Components

The directional flow loss components $\lambda_{\text{1-to-2}}$ and $\lambda_{\text{2-to-1}}$ control the strength of the flow loss in the 1-to-2 and 2-to-1 directions respectively. In our experiments, though the performance is not significantly different, we recommend using bidirectional flow loss for symmetry and stability.

 https://doi.org/10.52202/085713-0571

G Validation of the Fréchet Image and Video Metrics

G.1 Protocol

To confirm that the three Fréchet distances introduced in §6.3 (static content, dynamics and frame-wise) behave as intended on our *Stochastic Moving MNIST* benchmark, we performed a series of controlled experiments. Code for reproducing the experiments is available at `https://github.com/nkiyohara/srvp-fd`.

G.1.1 Datasets.

For each trial, two datasets were instantiated as follows:

- Content: Each containing 256 sequences of 25 frames.
- Production method: Same as in the main text with identical hyperparameters, varying only specific factors (digit classes, handwriting samples, or dynamics).
- Digit ordering: To remove ambiguity, a second copy of the first dataset was generated with digit indices swapped. The version yielding the smaller static Fréchet distance was retained.[1]

G.1.2 Conditions.

Four experimental conditions were repeated over 32 trials (each with a fresh random seed):

- **Different dynamics**: same digits and handwriting, different dynamics.
- **Different digits**: different digit classes, identical dynamics.
- **Different handwriting**: same digit classes, different handwriting, identical dynamics.
- **Different both**: different digit classes *and* different dynamics.

G.1.3 Computation of distances.

Distance computation details:

- Encoder: Public SRVP encoder of Franceschi et al. [14].
- Frame-wise scores: Averaged the 25 frame distances to yield a single number per sequence pair.

G.2 Results

Table 16 reports the mean and standard deviation (over the 32 trials) of each metric. The metrics reflect the intended factors:

- Varying only the dynamics (*different dynamics*) drives the **dynamics** distance up by an order of magnitude while leaving the static distance low.
- Altering appearance while keeping motion fixed (*different digits*/*handwriting*) raises the **static** distance, with digits $>$ handwriting as expected, and leaves the dynamics distance low.
- When both factors change (*different both*), *both* metrics are simultaneously high.
- Frame-wise distances stay near zero unless both content and motion differ, indicating that they act as a weak sanity check.

[1] Without this post-hoc swap the static metric can be artificially inflated when the same digits appear in reverse order.

Table 13: Effect of flow loss weight λ on test MSE (mean $\pm$ 95% confidence interval in t-statistic) for CMU Motion Capture (Setup 2). Auxiliary inner steps are fixed to $K = 3$. Flow loss weight $\lambda = 1.0$, bidirectional($\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1.0$), and KL weight $\beta = \beta_{\text{skip}} = 0.3$.

Flow loss weight (λ)	MSE
0.001	4.08 ± 0.46
0.003	3.77 ± 0.37
0.01	3.73 ± 0.48
0.03	3.36 ± 0.25
0.1	3.43 ± 0.29
0.3	3.60 ± 0.33
1.0	3.41 ± 0.27

Table 14: Effect of directional flow loss components on test MSE (mean $\pm$ 95% confidence interval in t-statistic) for CMU Motion Capture (Setup 2). Flow loss weight $\lambda = 1.0$, inner loop steps for bridge model $K = 3$, and KL weight $\beta = \beta_{\text{skip}} = 0.3$.

Flow loss direction	MSE
1-to-2 only $\lambda_{\text{1-to-2}} = 1, \lambda_{\text{2-to-1}} = 0$	3.66 ± 0.50
2-to-1 only $\lambda_{\text{1-to-2}} = 0, \lambda_{\text{2-to-1}} = 1$	3.48 ± 0.46
Bidirectional $\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1$	3.41 ± 0.27

Table 15: Effect of auxiliary optimisation steps K on test MSE (mean $\pm$ 95% confidence interval in t-statistic) for CMU Motion Capture (Setup 2). Flow loss weight $\lambda = 1.0$, bidirectional($\lambda_{\text{1-to-2}} = \lambda_{\text{2-to-1}} = 1.0$), and KL weight $\beta = \beta_{\text{skip}} = 0.3$. "Simultaneous" refers to updating auxiliary and main models jointly without inner loops for bridge model, as described in Appendix C.2.

K (auxiliary inner steps)	MSE
0 (no auxiliary training)	261.82 ± 124.57
0 (simultaneous)	3.59 ± 0.61
1	3.35 ± 0.39
3 (default)	3.41 ± 0.27
5	3.68 ± 0.40
10	3.59 ± 0.38

Table 16: Mean $\pm$ standard deviation of the three Fréchet distances across 32 trials.

Condition	**Static**	**Dynamics**	**Frame mean**
Different dynamics	2.57 ± 1.40	14.85 ± 5.48	0.80 ± 0.14
Different digits	5.66 ± 1.82	1.75 ± 1.83	0.13 ± 0.06
Different handwriting	3.74 ± 1.65	1.79 ± 1.66	0.09 ± 0.05
Different both	6.78 ± 1.95	14.74 ± 5.79	0.87 ± 0.13

https://doi.org/10.52202/085713-0571

G.3 Discussion

The clear separation between conditions, the low variance within each condition and the alignment with human intuition together demonstrate that our Fréchet metrics are *both reliable and interpretable*. They therefore provide a sound basis for quantitative evaluation of generative video models on Stochastic Moving MNIST, capturing complementary aspects of visual content and motion without leakage between the two.