Generative Models for Generic Data

Andrew Campbell

St Peter's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2025

Acknowledgements

First and foremost I would like to thank my advisors Arnaud and Tom. I greatly appreciate their belief in me and unwavering support. There were countless times during my PhD where I may have been at a loss as to what to do but our meetings always gave me a new confidence and excitement for research.

I would like to thank all the friends made over these past 4 years for making this such a great experience, among them are Adam F, Adam G, Amitis, Angus, Anna, Bobby, Carlo, Chris, Dan, Desi, Emi, Emile, Faaiz, Fabian, Freddie, Guneet, Gonzalo, Jakiw, James, Jannik, Jef, Jin, Justin, Kamélia, Leo, Marcus, Martin, Max, Michael, Rob, Robert, Sahra, Saif, Sam, Shahine, Thu, Tim, Tyler, Valentin, Vik and Yuyang. I'd also like to thank the department staff for all their help, especially Frédérique, Joanna, Jonathan, Mark and Stuart for always helping me out with any question I had.

I owe a huge debt of gratitude to the wonderful collaborators I have had the opportunity to work including Vincent, Wenlong, Yuyang, Joe, Valentin, Will, Christian, Jason, Emile, Andrew, Chris, Saif and Sulin. I have learnt so much from working together and this thesis would not have been possible without them.

I am also truly indebted to my internship mentors Adi at Tiktok, José at MSR and Arnaud at GDM for giving me the opportunity to work on exciting problems and welcoming me to the team. I have fond memories of the lunches and table tennis with the AI4Science team in Cambridge and the regular breakfasts and lunches with friends at GDM.

I owe a big thank you to Jason and Ignacio for bringing about my research visit and Tommi and Regina for hosting me. I learned a great deal from my time in Cambridge and would like to thank everyone at CSAIL who made me feel welcomed including Bowen, Gabri, Hannes, Itamar, Jakub, Jeremy, Mingyu, Peter H, Peter M, Sean, Sulin, Tally, Timur, Xiang and Yilun.

Finally, and most dear to me, I must give my deepest thanks to my mum Gill, my dad Paul and my sister Charlotte. Without your constant love and support I would not have been able to reach the end of this journey. Thank you for always being there for me.

Abstract

Generative models parameterized through large neural networks are now ubiquitous across machine learning. These models are able to generate highly realistic samples, ranging from human-like text to novel chemical compounds. With the ever growing list of potential application areas comes the need to continually adapt generative models to the complex data types that appear in the real world. This thesis aims to endow generative models with the ability to operate over generic data, requiring our generative frameworks to be general enough to handle any structures and regularities that may be present in the dataset. We focus on the subfamily of generative models that create data by simulating an iterative generative process which have become the best performing model class across a range of tasks due to their simple training procedure and high quality samples. This brings new challenges for generic data modelling because appropriate generative processes must be designed to operate on complex data spaces.

We begin by developing a method for generating discrete data through simulating continuous time Markov chains (CTMCs). Our approach translates techniques from continuous space diffusion models parameterized by stochastic differential equations into discrete space. We then turn our attention to data that can vary in dimensionality by giving continuous space diffusion models the ability to add dimensions as needed during generation. In the second half of the thesis, we switch focus to flow-based generative models which offer a simple alternative approach to creating generative processes. We first apply flows operating on the space of rigid body motions to the task of protein motif-scaffolding. We then expand flow-based modelling to discrete data and multi-modal data by utilizing CTMCs in combination with continuous space flows. We apply our method to protein co-design, simultaneously generating the continuous protein 3D structure and corresponding discrete amino acid sequence.

Contents

1 Introduction

Contents

Generative modelling is the act of taking a dataset of observed samples and then creating new samples that are in some way similar to those examples in the dataset but are not exact replicas. The generative modelling problem is underspecified

because we have no rigorous definition of the way in which we desire samples to be 'similar' to the dataset and no definition of how different we need the new samples to be from those examples we already have. The success criteria, in the end, come down to the specific application. For example, when generating novel protein structures, we desire the samples to be similar to the observations in the sense that they are thermodynamically stable and utilize folding mechanisms that are physically plausible but different in the sense that they are novel combinations of structural motifs or achieve folds not seen in nature. Any useful generative model needs to achieve a balance between these two counteracting requirements.

In this chapter, we explore the concept of generative modelling as a whole. We start by considering why we need to go to such lengths in the first place to construct elaborate probabilistic models to generate data. Utilizing this intuition, we detail 'classical' approaches to generative modelling, specifically Variational Autoencoders (VAEs) (Kingma et al. 2013) and Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) and see how they approach the generative modelling problem. By understanding the limitations of these classical approaches in terms of sample quality, training stability and mode coverage we can motivate the use of generative processes as a technique to generate data.

We then introduce the broad framework of generative processes. We aim to provide a unifying perspective encompassing diffusion and flow-based models on continuous and discrete data. We postpone a detailed literature review of these methods until Chapter 2. Here, we will provide an introduction to some core stochastic process concepts, importantly including continuity equations on continuous and discrete spaces. We will utilize continuity equations to unify the approaches and discuss the training of generative process methods.

1.1 Why is Generative Modelling Difficult?

Assume we are given a dataset of samples $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$ from some underlying data distribution $p_{\text{data}}(\mathbf{x})$. Our task is to generate new samples from p_{data}. A first attempt at creating a generative model for this task could be to approximate p_{data} with the 'empirical data distribution', $\hat{p}_{\text{data}}$, that is the distribution given by the samples,

$$\hat{p}_{\text{data}}(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\delta(\mathbf{x} = \mathbf{x}_n) \tag{1.1}$$

where $\delta(\mathbf{x} = \mathbf{x}_n)$ is a Dirac delta distribution centred at $\mathbf{x}_n$. We do generative modelling by sampling from $\hat{p}_{\text{data}}$. Unfortunately, this is not very useful because sampling from $\hat{p}_{\text{data}}$ simply recapitulates samples from the training dataset. In order to proceed, we need to make some use of a neural network that learns generalizable features from $\mathcal{D}$ and can recombine those features in new ways to make new samples.

To train this neural network, we need to come up with some learning objective. We are not in a supervised learning setting - there are no input-output pairs to learn from. We only have access to the empirical data distribution $\hat{p}_{\text{data}}$ formed by our dataset of samples. Therefore, the most simple loss to try would be to take a neural network with inputs all set to 0 and have it try and predict data points.

$$\min_{\theta} \quad \mathbb{E}_{\hat{p}_{\text{data}}(\mathbf{x})}\left[||g_\theta(\mathbf{0}) - \mathbf{x}||^2\right] \tag{1.2}$$

where g_θ is our neural network with parameters θ and we use $\mathbb{E}_{\hat{p}_{\text{data}}}$ to refer to the expectation taken with respect to samples from our training set $\mathcal{D}$. Of course, this learning method will fail because it can be easily proven that the optimal solution to Eq. 1.2 is the mean $g_{\theta^*}(\mathbf{0}) = \mathbb{E}_{\hat{p}_{\text{data}}(\mathbf{x})}[\mathbf{x}]$. A network that simply outputs the dataset mean is not useful for generative modelling.

To remedy the issue in the prior approach, we can try to artificially create input-output pairs for training the neural network. For lack of a better alternative, we could assign a random vector, $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$, to each sample in the training set

to create an augmented dataset, $\mathcal{D}^{\text{aug}} = \{\mathbf{x}_n, \epsilon_n\}_{n=1}^N$. The corresponding empirical data distribution becomes

$$\hat{p}_{\text{data}}^{\text{aug}}(\mathbf{x}, \epsilon) = \frac{1}{N}\sum_{n=1}^{N} \delta(\mathbf{x} = \mathbf{x}_n)\delta(\epsilon = \epsilon_n) \tag{1.3}$$

We then train our neural network to predict $\mathbf{x}$ given its corresponding ϵ.

$$\min_{\theta} \quad \mathbb{E}_{\hat{p}_{\text{data}}^{\text{aug}}(\mathbf{x},\epsilon)}\left[||g_\theta(\epsilon) - \mathbf{x}||^2\right] \tag{1.4}$$

The solution to this learning problem is a neural network g_{θ^*} that predicts the training set datapoint corresponding to the noise vector that was assigned to it, $g_{\theta^*}(\epsilon_n) = \mathbf{x}_n$. If we were to sample a fresh $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$ and input this into g_{θ^*}, the network would output something novel, however it would be unlikely that the sample would resemble the training dataset because there was no structure imparted into the $\epsilon \rightarrow \mathbf{x}$ mapping during the construction of our training set. When the mapping is completely arbitrary, the neural network simply memorizes the training set and this does not induce generalization.

This can be demonstrated on the MNIST dataset of handwritten digits. A small network using the U-Net architecture (Ronneberger et al. 2015) was trained using the objective given by Eq 1.4. We show the outputs of the neural network $g_\theta(\epsilon)$ in Figure 1.1. We can see that for ϵ inputs that are seen during training paired with a data sample $\mathbf{x}_n$, the neural network is able to almost exactly reproduce the corresponding $\mathbf{x}_n$ datapoint. However, when sampling fresh $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$ the neural network entirely fails to produce realistic data samples.

To fix this you could try to randomize the ϵ sample that gets assigned to each $\mathbf{x}$ giving a training joint distribution of

$$\hat{p}_{\text{data}}^{\text{fresh}}(\mathbf{x}, \epsilon) = \frac{1}{N}\sum_{n=1}^{N} \delta(\mathbf{x} = \mathbf{x}_n)\mathcal{N}(\epsilon; 0, I) = \hat{p}_{\text{data}}(\mathbf{x})\mathcal{N}(\epsilon; 0, I) \tag{1.5}$$

However, since ϵ and $\mathbf{x}$ become completely uncoupled in $\hat{p}_{\text{data}}^{\text{fresh}}$, one can again prove the optimal network outputs the dataset mean $g_{\theta^*}(\epsilon) = \mathbb{E}_{\hat{p}_{\text{data}}(\mathbf{x})}[\mathbf{x}]$.

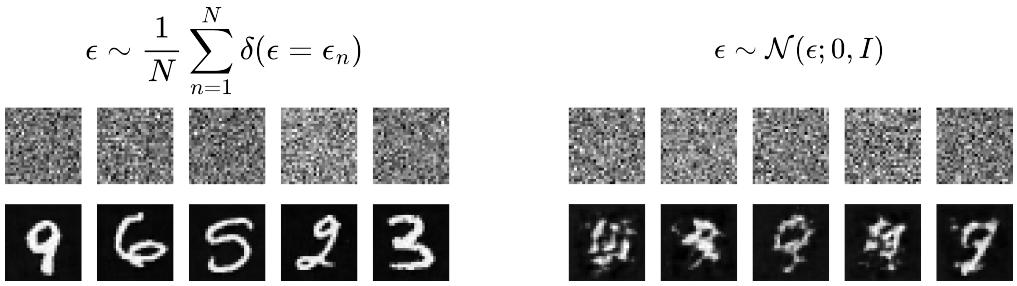


Figure 1.1: Top row ϵ inputs into the neural network. **Bottom row** $g_\theta(\epsilon)$ outputs from the neural network. **Left Block** ϵ inputs are taken from the empirical distribution used during training. **Right Block** fresh ϵ values are sampled from $\mathcal{N}(\epsilon; 0, I)$.

These previous examples show the difficulty in developing a working generative model. Classical generative models overcome these limitations from two angles. The first is to come up with a better way to assign neural network inputs ϵ_n to training datapoints $\mathbf{x}_n$ so that new datapoints not seen in the dataset can be created. This line of attack leads to Variational Autoencoders. The second is to try to loosen the training objective so that we do not have specific targets for the network output for each input but we instead try to match the *distribution* of network outputs to the *distribution* of the training dataset. This line of attack leads to Generative Adversarial Networks.

1.2 Variational Autoencoders

Variational Autoencoders (VAEs) (Kingma et al. 2013) can be thought of as a way to create better ϵ - $\mathbf{x}$ pairs for training the generative network with. The idea is to introduce a second *encoder* neural network, $E_\phi(\mathbf{x})$, that gives us an ϵ value associated to each $\mathbf{x}$. We then train the generative network, termed the *decoder*, $D_\theta(\epsilon)$, that maps from ϵ back to $\mathbf{x}$. To sample new datapoints, we create a fresh ϵ that is not associated to any $\mathbf{x}$ in the training dataset, push it though the decoder and hopefully obtain a novel datapoint.

Given just this motivation, naively one could train E_ϕ and D_θ by simply minimizing a cyclic reconstruction loss whereby we take a training datapoint, encode

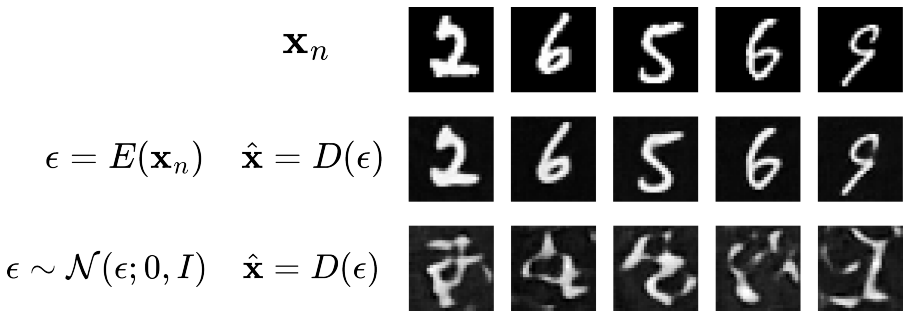


Figure 1.2: Simple encoder-decoder generative model trained using Eq 1.6. **Top row** Training datapoints. **Middle row** Reconstructions of the datapoints by passing them through the encoder and then the decoder. **Bottom row** Freshly sampled datapoints through first sampling new latent vectors ϵ and then passing these through the decoder.

it, decode it and then compare this decoded sample to the original datapoint.

$$\min_{\theta,\phi} \quad \mathbb{E}_{\hat{p}_{\text{data}}(\mathbf{x})}\left[||D_\theta(E_\phi(\mathbf{x})) - \mathbf{x}||^2\right] \tag{1.6}$$

After training with this loss, to generate new datapoints, we could try creating fresh latent vectors that are not associated to any existing training datapoint and pushing these through the decoder. For simplicity, we could create new ϵ by sampling a standard normal,

$$\epsilon \sim \mathcal{N}(\epsilon; 0, I), \qquad \hat{\mathbf{x}} = D_\theta(\epsilon). \tag{1.7}$$

Figure 1.2 shows the result of this approach on MNIST. We can see that the model can very successfully reproduce training examples, however, it entirely fails to produce realistic looking novel samples when using fresh latent vectors $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$.

The problem with this approach is that the distribution of latent vectors that the decoder sees during training, $\int_{\mathbf{x}} \hat{p}_{\text{data}}(\mathbf{x})\delta(\epsilon - E_\phi(\mathbf{x}))\mathrm{d}\mathbf{x}$, can be entirely different to the distribution of latent vectors we use at test time to sample novel datapoints $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$. VAEs seek to remedy this by adding additional regularization to the training latent vector distribution so that it matches the distribution we will sample from at test time.

VAEs are defined as a probabilistic model. In the generative direction, we have a decoding distribution $p_\theta(\mathbf{x}|\epsilon)$ typically parameterized using a a neural network

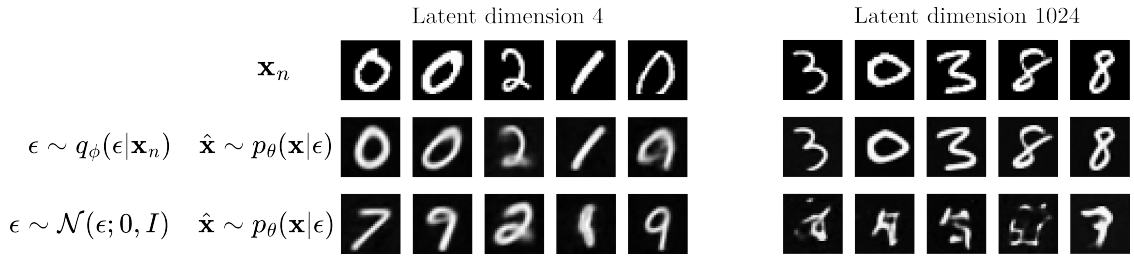


Figure 1.3: VAE generative models trained with varying latent vector ϵ dimensionality. The dimensionality 4 model is shown on the left and the dimensionality 1024 model is shown on the right. **Top Row** Training datapoints. **Middle Row** Reconstructions of training datapoints by first sampling the encoder $\epsilon \sim q_\phi(\epsilon|\mathbf{x}_n)$ and then sampling the decoder $\hat{\mathbf{x}} \sim p_\theta(\mathbf{x}|\epsilon)$. **Bottom Row** Novel synthetic datapoints created by first sampling ϵ from the prior $\mathcal{N}(\epsilon; 0, I)$ and then sampling the decoder $\hat{\mathbf{x}} \sim p_\theta(\mathbf{x}|\epsilon)$.

that gives the mean of a Gaussian distribution with fixed variance $p_\theta(\mathbf{x}|\epsilon) = \mathcal{N}(\mathbf{x}; D_\theta(\epsilon), \sigma_x^2 I)$. This decoding distribution can be easily modified for other data types as we discuss later in Section 2.4.1. We also maintain an encoding distribution, again a Gaussian distribution parameterized using a neural network $q_\phi(\epsilon|\mathbf{x}) = \mathcal{N}(\epsilon; E_\phi^\mu(\mathbf{x}), E_\phi^\sigma(\mathbf{x})^2)$. We use $p(\epsilon) = \mathcal{N}(\epsilon; 0, I)$ as the test time distribution over fresh latent vectors. p_θ and q_ϕ are trained jointly by maximizing a lower bound on the model's log-likelihood assigned to the training datapoints.

$$\log p_\theta(\mathbf{x}) = \log \mathbb{E}_{p(\epsilon)} \left[p_\theta(\mathbf{x}|\epsilon) \right] \tag{1.8}$$

$$= \log \mathbb{E}_{q_\phi(\epsilon|\mathbf{x})} \left[\frac{p(\epsilon)}{q_\phi(\epsilon|\mathbf{x})} p_\theta(\mathbf{x}|\epsilon) \right] \tag{1.9}$$

$$\geq \mathbb{E}_{q_\phi(\epsilon|\mathbf{x})} \left[\log \frac{p(\epsilon) p_\theta(\mathbf{x}|\epsilon)}{q_\phi(\epsilon|\mathbf{x})} \right] \tag{1.10}$$

$$= \mathbb{E}_{q_\phi(\epsilon|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\epsilon) \right] - \mathrm{KL}(q_\phi(\epsilon|\mathbf{x})||p(\epsilon)) \tag{1.11}$$

where KL is the Kullback-Leibler divergence. Eq. 1.11 is analogous to a regularized form of Eq 1.6 whereby $\mathbb{E}_{q_\phi(\epsilon|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\epsilon) \right]$ penalizes the cyclic reconstruction loss and $\mathrm{KL}\left(q_\phi(\epsilon|\mathbf{x})||p(\epsilon) \right)$ regularizes the encoder to match the test distribution. We trained a small network using Eq. 1.11 for $\epsilon \in \mathbb{R}^4$ and for $\epsilon \in \mathbb{R}^{1024}$. The results are shown in Figure 1.3.

When $\epsilon \in \mathbb{R}^4$, we see that the model generates semi-realistic novel datapoints however the samples look over-smoothed. This can be diagnosed by examining datapoint reconstructions which are also smooth suggesting the model is unable to store enough information in the latent vector ϵ to reconstruct the datapoint. To counter this, we could increase the ϵ dimensionality to a large value e.g. 1024. This correspondingly improves the reconstructions, however, now the generative model entirely fails to generate novel samples. To understand the issue, we can consider the aggregate posterior, $p_\phi^{\text{agg}}(\epsilon) = \int_{\mathbf{x}} \hat{p}_{\text{data}}(\mathbf{x}) q_\phi(\epsilon|\mathbf{x}) \mathrm{d}\mathbf{x}$ which is the distribution over ϵ that the decoder sees during training. In the high dimensional case, $p_\phi^{\text{agg}}(\epsilon)$ can be far from $p(\epsilon)$ creating a train-test shift and poor sample quality. This inherent trade-off between over-smoothed samples and train-test mismatch prevents VAEs achieving all of our generative model desiderata.

1.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) tackle the generative modelling problem by using a distributional loss that does not require the direct specification of ϵ - $\mathbf{x}$ pairs. The generative network or 'Generator' is written as $G_\theta(\epsilon)$ and is applied to latent vectors $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$. We aim to minimize the discrepancy between the generated distribution $p_g(\mathbf{x}) = \int_\epsilon p(\epsilon)\delta(\mathbf{x} - G_\theta(\epsilon))\mathrm{d}\epsilon$ and $p_{\text{data}}(\mathbf{x})$. The objective chosen in Goodfellow et al. 2014 to minimize is

$$\mathcal{L}_{\text{GAN}} = 2\text{JSD}\left(p_{\text{data}} || p_g\right) - \log 4 \tag{1.12}$$

where JSD is the Jensen-Shannon Divergence defined as

$$\text{JSD}\left(p_{\text{data}} \middle|\middle| p_g\right) := \frac{1}{2}\text{KL}\left(p_{\text{data}} \middle|\middle| \frac{1}{2}(p_{\text{data}} + p_g)\right) + \frac{1}{2}\text{KL}\left(p_g \middle|\middle| \frac{1}{2}(p_{\text{data}} + p_g)\right). \tag{1.13}$$

It can be shown that $\mathcal{L}_{\text{GAN}}$ can be written in terms of a classifier $p(Y = 1|\mathbf{x})$ that predicts if a given sample $\mathbf{x}$ comes from either p_{data} (Y = 1) or p_g (Y = 0).

$$\mathcal{L}_{\text{GAN}} = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\log p(Y = 1|\mathbf{x})\right] + \mathbb{E}_{p_g(\mathbf{x})}\left[\log\left(1 - p(Y = 1|\mathbf{x})\right)\right] \tag{1.14}$$

$p(Y = 1|\mathbf{x})$ is approximated using a second neural network, the discriminator $D_\phi(\mathbf{x})$. It turns out Eq. 1.14 can also be used as a maximum likelihood objective for $D_\phi(\mathbf{x})$, enabling the simultaneous training of G_θ and D_ϕ by minimax optimization,

$$\min_\theta \max_\phi \quad \mathcal{V}_{\text{GAN}} = \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{p(\epsilon)} \left[\log \left(1 - D_\phi(G_\theta(\epsilon))\right)\right]. \tag{1.15}$$

1.3.1 Training Difficulties

The minimax optimization procedure requires careful balancing between the optimization of the generator and the discriminator. To understand this issue, we can examine the gradient that is used to update the generator parameters. Suppose the discriminator $D_\phi(\mathbf{x})$ is parameterized by applying a sigmoid non-linearity to the output of the neural network. Letting $D_\phi^{\text{pre}}(\mathbf{x})$ be the pre-sigmoid activation, we have $D_\phi(\mathbf{x}) = \sigma(D_\phi^{\text{pre}}(\mathbf{x}))$ with $\sigma(z) = \frac{1}{1+e^{-z}}$. The generator gradient is then

$$\begin{aligned}
\nabla_\theta \mathcal{V}_{\text{GAN}} &= \mathbb{E}_{p(\epsilon)} \left[\nabla_\theta \log \left(1 - D_\phi(G_\theta(\epsilon))\right)\right] && (1.16)\\
&= \mathbb{E}_{p(\epsilon)} \left[\nabla_\theta \log \left(1 - \sigma(D_\phi^{\text{pre}}(G_\theta(\epsilon)))\right)\right] && (1.17)\\
&= \mathbb{E}_{p(\epsilon)} \left[-\sigma(D_\phi^{\text{pre}}(G_\theta(\epsilon))) \times \nabla_\theta D_\phi^{\text{pre}}(G_\theta(\epsilon))\right] && (1.18)\\
&= \mathbb{E}_{p(\epsilon)} \left[-D_\phi(G_\theta(\epsilon)) \times \nabla_\theta D_\phi^{\text{pre}}(G_\theta(\epsilon))\right] && (1.19)
\end{aligned}$$

We see that the discriminator's estimated probability that $G_\theta(\epsilon)$ is a data sample, $D_\phi(G_\theta(\epsilon))$, modulates the magnitude of the overall $\nabla_\theta \mathcal{V}_{\text{GAN}}$ gradient. This is an issue when the discriminator is very well trained and can confidently differentiate between generated and real samples. In this case we will have $D_\phi(G_\theta(\epsilon)) \approx 0$ resulting in a poor gradient signal for training the generator. To avoid this problem, the learning rate and number of update steps for the discriminator needs to be carefully controlled so that it is not able to confidently distinguish between real and generated samples during the entire training run.

Furthermore, we see that the gradient signal for updating the generator parameters θ flows through the discriminator network, $\nabla_\theta D_\phi^{\text{pre}}(G_\theta(\epsilon))$. This can cause issues when the input-output relationship for the discriminator $D_\phi(\mathbf{x})$ does not produce a

useful gradient signal for updating the generator. We would like $\nabla_{\mathbf{x}} D_\phi(\mathbf{x})$ to point in the direction towards real samples. However, this is not a given, especially when the Lipschitz constant of D_ϕ is large. In this case, there can be regions of $\mathbf{x}$ space that have very little change in D_ϕ value with respect to $\mathbf{x}$ and some regions where D_ϕ changes very quickly. This is not a well conditioned landscape for optimization. Wasserstein GANs (Arjovsky et al. 2017; Gulrajani et al. 2017) go some way to fixing this issue by regularizing the Lipschitz constant of the discriminator network.

1.3.2 Mode Collapse

A further common issue with GANs is their tendency to drop modes in the data distribution, thereby 'collapsing' onto the remaining modes. This problem originates from the divergence measure used to derive the GAN objective (Eq. 1.12). To understand the issue, let us first consider a mode-covering divergence: the forward Kullback-Leibler divergence.

$$\text{Forward Kullback-Leibler} \qquad \mathrm{KL}\left(p_{\text{data}} || p_g\right) = \mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_g(\mathbf{x})}\right] \tag{1.20}$$

We say the forward KL is mode-covering because it includes an infinite penalty if there are regions of $\mathbf{x}$ space for which p_{data} places mass but the generated distribution p_g does not. To see this, we consider the case of sampling an $\mathbf{x}$ in the $p_{\text{data}}(\mathbf{x}) > 0, p_g(\mathbf{x}) \to 0$ region and then evaluating $\log \frac{p_{\text{data}}(\mathbf{x})}{p_g(\mathbf{x})}$. As $p_g(\mathbf{x}) \to 0$, the divergence diverges, applying an infinite penalty. Now, we can compare this behaviour to the Jensen-Shannon divergence that a GAN is minimizing.

$$\mathrm{JSD}\left(p_{\text{data}} \middle|\middle| p_g\right) = \frac{1}{2}\mathrm{KL}\left(p_{\text{data}} \middle|\middle| \frac{1}{2}(p_{\text{data}} + p_g)\right) + \frac{1}{2}\mathrm{KL}\left(p_g \middle|\middle| \frac{1}{2}(p_{\text{data}} + p_g)\right) \tag{1.21}$$

$$= \frac{1}{2}\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{1}{2}\left(p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})\right)}\right] + \tag{1.22}$$

$$\frac{1}{2}\mathbb{E}_{p_g(\mathbf{x})}\left[\log \frac{p_g(\mathbf{x})}{\frac{1}{2}\left(p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})\right)}\right] \tag{1.23}$$

Now, for the first term, consider the same case where we sample $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ in a region over which the generated distribution $p_g(\mathbf{x})$ places no mass. The divergence contribution in this case is

$$\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{1}{2}\left(p_{\text{data}}(\mathbf{x}) + 0\right)} = \log 2. \tag{1.24}$$

Therefore, the JSD does not applying an infinite penalty when the generated distribution drops modes of the data distribution.

At a high level, the fundamental aspects of a mode covering objective are that we need to first sample $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$, evaluate $p_g(\mathbf{x})$ and then apply a diverging penalty as $p_g(\mathbf{x}) \to 0$. This requires the ability to 'invert' the generative model. Rather than always sampling an input $\epsilon \sim p(\epsilon)$ and running this forward through the generative network, we need to be able to take a desired output, $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ and work backwards through the network to assess what volume of the input space corresponds to generating this given $\mathbf{x}$. If the measure of this volume under $p(\epsilon)$ is vanishingly small then our generative model will never generate the given $\mathbf{x}$ and so we can apply an infinite penalty. Unfortunately, this is intractable to do in the standard GAN setup. In comparison, VAEs are trained with a mode-covering maximum likelihood objective which is achieved by using the posterior $q_\phi(\epsilon|\mathbf{x})$ as the inversion tool.

Due to their mode-seeking behaviour, GANs tend to produce high quality samples but due to issues relating to mode coverage and unstable training they are far from a perfect generative modelling paradigm.

1.4 Multi-Stage Generative Models

The previously described generative models generate data in a single step, i.e. in one forward pass of the neural network. Improvements have been found by stacking multiple generative layers together which splits the overall generative modelling task of going from $\epsilon \sim p(\epsilon)$ to $\mathbf{x} \sim p_{\text{data}}$ into more manageable chunks.

1.4.1 Hierarchical VAEs

A hierarchical VAE (Vahdat et al. 2020; Tomczak et al. 2018; Klushyn et al. 2019), is defined by splitting the latent variable up into L chunks, $\epsilon = [\epsilon_1, \ldots, \epsilon_L]$. The

prior is then defined as an autoregressive distribution over ϵ,

$$p(\epsilon) = \prod_{l=1}^{L} p(\epsilon_l|\epsilon_{1:l-1}). \tag{1.25}$$

The posterior also splits autoregressively,

$$q_\phi(\epsilon|\mathbf{x}) = \prod_{l=1}^{L} q_\phi(\epsilon_l|\epsilon_{1:l-1}, \mathbf{x}). \tag{1.26}$$

In this case, the log-likelihood lower-bound becomes

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\epsilon|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\epsilon)\right] - \mathrm{KL}\left(q_\phi(\epsilon|\mathbf{x})||p(\epsilon)\right) \tag{1.27}$$

$$= \mathbb{E}_{q_\phi(\epsilon|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\epsilon)\right] - \sum_{l=1}^{L} \mathbb{E}_{q_\phi(\epsilon_{1:l-1}|\mathbf{x})}\left[\mathrm{KL}\left(q_\phi(\epsilon_l|\epsilon_{1:l-1}, \mathbf{x})||p(\epsilon_l|\epsilon_{1:l-1})\right)\right]. \tag{1.28}$$

The splitting of the latent variable into L components with an autoregressive posterior allows the posterior to be much more expressive and results in the ability to store more information about $\mathbf{x}$ into the latent state whilst still ensuring the aggregate posterior matches the prior. However, the large stack of parameterized distributions is difficult to train jointly as the gradient must propagate through all layers during the calculation of the log-likelihood lower bound thus being susceptible to vanishing or exploding gradients. Vahdat et al. 2020 attempt to control this through spectral normalization of the neural network weights.

1.4.2 Progressive GANs

A similar observation was made in the training of GANs, in that splitting the generative modelling task into smaller subtasks can improve performance. Karras et al. 2018, for example, introduce the progressive growing of GANs where training starts with training at low resolutions and then proceeds to higher resolutions only once the initial training stage has had the chance to converge. This improved training stability and sample quality.

1.4.3 Normalizing Flows

Normalizing flows (Rezende et al. 2015; Dinh et al. 2015) are an alternative approach to multi-step generative models that stack together a series of invertible transformations, $\mathbf{x} = f_K \circ \cdots \circ f_2 \circ f_1(\epsilon_0)$ with parameters θ. This transforms an initial prior distribution $p(\epsilon_0)$ into a complex output distribution $p_\theta(\mathbf{x})$, the log-likelihood of which can be calculated through an iterated change of variables formula

$$\log p_\theta(\mathbf{x}) = \log p(\epsilon_0) - \sum_{k=1}^{K} \log \left| \det \frac{\partial f_k}{\partial \epsilon_{k-1}} \right| \tag{1.29}$$

This allows the use of mode-covering maximum likelihood training but also requires the use of specialized network architectures that are invertible and have tractable determinants. The idea can be extended to continuous time ordinary differential equations (ODEs) (Chen et al. 2018a) which track the probability distribution shift through integrating an instantaneous change of variables formula.

1.5 From Generative Models to Generative Processes

Multi-stage generative models go some way towards solving the issues of the single step models presented in Sections 1.2 & 1.3 through splitting up the generative task into smaller pieces. However, multi-stage models face issues with training because the training gradients must propagate through the entire generative stack. This necessitates the use of spectral normalization for hierarchical VAEs, progressive training in GANs and ODE solvers running at training time for normalizing flows.

It is in this landscape that generative processes were developed which parameterize the generative model through a process e.g. Markov chain, ODE or SDE that generates data. Rather than simply learning the entire process from scratch, the key insight of generative processes is that we must impart some prior structure onto the mechanism through which data is generated. This structure, in a sense, 'guides' the model towards learning a specific desired behaviour during the generating

process. Explicitly, diffusion models desire the generative process to be the time reversal of a pre-specified corruption process that destroys information in the data. Flow-based models, on the other hand, desire the generative process to be a mixture of conditional processes, each of which would converge towards a specific datapoint.

These structures remove the need to backpropagate through the entire generative process during learning. The model can instead be learned by taking intermediate points in the process in isolation leading to stable training. The prior imparted structure has the additional benefit of providing a form of inversion which results in the learned models being mode-covering. To fully understand and make explicit these intuitions, we must first introduce concepts from stochastic process theory.

1.6 Stochastic Processes

1.6.1 Stochastic Differential Equations

Stochastic Differential Equations (SDEs) are a commonly used class of stochastic process for continuous data, $\mathbf{x} \in \mathbb{R}^n$. An SDE is defined through a time dependent drift term $f_t(\mathbf{x}) : \mathbb{R}^n, \mathbb{R} \rightarrow \mathbb{R}^n$ and a time dependent diffusion coefficient $g_t : \mathbb{R} \rightarrow \mathbb{R}$. We can write the dynamics of an SDE in the following differential form (Särkkä et al. 2019),

$$\mathrm{d}\mathbf{x} = f_t(\mathbf{x})\mathrm{d}t + g_t \mathrm{d}w_t \tag{1.30}$$

where $\mathrm{d}w_t$ is a Brownian motion increment. The dynamics can be understood intuitively as that in the next infinitesimal update step $\mathrm{d}t$, the state moves a small increment in the direction of the drift, $f_t(\mathbf{x})\mathrm{d}t$ and a small amount of noise is added $g_t \mathrm{d}w_t$, with the level of noise dictated by g_t. There are multiple ways to define Brownian motion, here we will use the approach of Del Moral et al. 2017. In infinitesimal timestep $\mathrm{d}t$, a particle undergoing 1D Brownian motion $\mathrm{d}x = g_t \mathrm{d}w_t$ will either move $+g_t\sqrt{\mathrm{d}t}$ or $-g_t\sqrt{\mathrm{d}t}$ each with probability $\frac{1}{2}$. In the multi-dimensional case, each dimension propagates as its own independent Brownian motion.

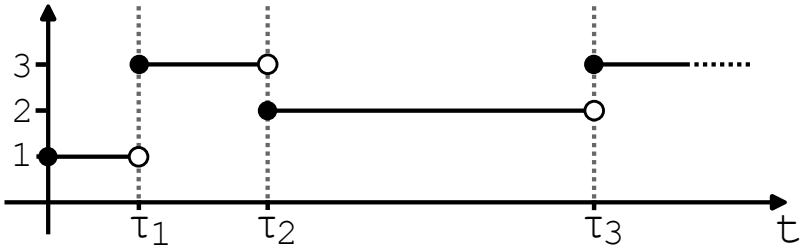


Figure 1.4: An example trajectory for a three state CTMC. Jump times are labelled as τ_1, τ_2, τ_3.

1.6.2 Continuous Time Markov Chains

For data living in a discrete space, $\mathbf{x} \in \{1, 2, \ldots, S\}^D$, we can define a stochastic process using a Continuous Time Markov Chain (CTMC). A particle following CTMC dynamics alternates between resting in its current state and jumping to a randomly chosen other state, Figure 1.4. A CTMC is defined by an initial distribution $p_0(\mathbf{x}_0)$ and a rate matrix $R_t : \mathbb{R} \to \mathbb{R}^{S^D \times S^D}$. The rate matrix determines both the frequency at which jumps occur during the trajectory and the destinations of those jumps. Within the next infinitesimal timestep $\mathrm{d}t$, if the current state is $\mathbf{x}_t$, then the probability that the particle will jump to state j is given by $R_t(\mathbf{x}_t, j)\mathrm{d}t$ where we have used $R_t(\mathbf{x}_t, j)$ to denote the $\mathbf{x}_t$'th row and j'th column of the rate matrix. This transition probability can therefore be written as

$$p_{t+\mathrm{d}t|t}(j|\mathbf{x}_t) = \begin{cases} R_t(\mathbf{x}_t, j)\mathrm{d}t & \text{for } j \neq \mathbf{x}_t \\ 1 - \sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j)\mathrm{d}t & \text{for } j = \mathbf{x}_t. \end{cases} \tag{1.31}$$

If we then make the definition that the diagonal entries of the rate matrix are equal to the negative row sum,

$$R_t(\mathbf{x}_t, \mathbf{x}_t) := -\sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j), \tag{1.32}$$

then this transition probability can be written succinctly as

$$p_{t+\mathrm{d}t|t}(j|\mathbf{x}_t) = \delta\{\mathbf{x}_t = j\} + R_t(\mathbf{x}_t, j)\mathrm{d}t. \tag{1.33}$$

where $\delta\{\mathbf{x}_t = j\}$ is a Kronecker delta function that is 1 when $\mathbf{x}_t = j$ and is 0 otherwise.

1.6.3 Fokker-Planck-Kolmogorov

A Fokker-Planck-Kolmogorov (FPK) equation can be understood as a continuity equation for probability mass. The rate of accumulation of probability mass in state $\mathbf{x}$, $\partial_t p_t(x)$, can be related to the incoming and outgoing flows of probability mass under the stochastic process dynamics. This general concept has different specific instantiations depending on the type of stochastic process under consideration. We here describe the FPK equation for SDEs and CTMCs.

FPK equation for SDEs

We will derive the FPK equation for the SDE defined in Eq. 1.30. The general proof technique is to consider an infinitesimal volume surrounding state $\mathbf{x}$. For ease of exposition, we will assume a 3-dimensional space, however, the results generalize to higher dimensions. Our infinitesimal volume has extent ν_1, ν_2, ν_3 in the 3 spatial dimensions for each $\nu_d \in \mathbb{R} << 1$. One corner of the cube has coordinate $\mathbf{x}$ whilst the 3 adjacent corners of the cube have coordinates $\mathbf{x} + \nu_1\mathbf{e}_1$, $\mathbf{x} + \nu_2\mathbf{e}_2$ and $\mathbf{x} + \nu_3\mathbf{e}_3$ respectively. $\mathbf{e}_d$ represents a unit vector pointing along the d-th dimension. For particles following the motion defined by Eq. 1.30, there are two mechanisms by which mass can flow into and out of an infinitesimal volume. The first is due to the drift and the second is due to the Brownian motion. We will first consider the contribution to the change in probability mass inside the volume due to the drift.

Drift Contribution Consider an infinitesimal timestep $\mathrm{d}t$. For the 1-st dimension, the probability mass that enters this side of the volume is

$$f_t^{(1)}(\mathbf{x})\mathrm{d}t\nu_2\nu_3 p_t(\mathbf{x}) \tag{1.34}$$

because $f_t^{(1)}(\mathbf{x})\mathrm{d}t$ gives us the total distance particles can travel in time $\mathrm{d}t$ in the 1 direction so multiplying this by $\nu_2\nu_3$ gives the volume within which particles will enter our infinitesimal volume, Finally, multiplying by $p_t(\mathbf{x})$ then results in

the total probability mass entering. Looking at the other side of the volume, the total mass exiting is

$$f_t^{(1)}(\mathbf{x}+\nu_1\mathbf{e}_1)\mathrm{d}t\nu_2\nu_3 p_t(\mathbf{x}+\nu_1\mathbf{e}_1) \tag{1.35}$$

$$= \left(\partial^{(1)} f_t^{(1)}(\mathbf{x})\nu_1 + f_t^{(1)}(\mathbf{x})\right)\mathrm{d}t\nu_2\nu_3\left(\partial^{(1)} p_t(\mathbf{x})\nu_1 + p_t(\mathbf{x})\right), \tag{1.36}$$

where we have used a first order Taylor expansion on both $f_t^{(1)}$ and p_t. We use $\partial^{(1)}$ to represent the partial spatial derivative in the 1 direction. The total profit in probability mass is therefore,

$$\text{profit} = f_t^{(1)}(\mathbf{x})\mathrm{d}t\nu_2\nu_3 p_t(\mathbf{x}) - \tag{1.37}$$

$$\left(\partial^{(1)} f_t^{(1)}(\mathbf{x})\nu_1 + f_t^{(1)}(\mathbf{x})\right)\mathrm{d}t\nu_2\nu_3\left(\partial^{(1)} p_t(\mathbf{x})\nu_1 + p_t(\mathbf{x})\right) \tag{1.38}$$

$$= -\partial^{(1)} f_t^{(1)}(\mathbf{x}) p_t(\mathbf{x})\mathrm{d}t\nu_1\nu_2\nu_3 - \tag{1.39}$$

$$f_t^{(1)}(\mathbf{x})\partial^{(1)} p_t(\mathbf{x})\mathrm{d}t\nu_1\nu_2\nu_3 + \text{higher order terms.} \tag{1.40}$$

This analysis can be repeated in the other dimensions giving a total profit of probability mass due to the drift, ignoring higher order terms, of

$$\text{total profit} = \mathrm{d}t\nu_1\nu_2\nu_3\Bigg(- \partial^{(1)} f_t^{(1)}(\mathbf{x}) p_t(\mathbf{x}) - f_t^{(1)}(\mathbf{x})\partial^{(1)} p_t(\mathbf{x}) \tag{1.41}$$

$$- \partial^{(2)} f_t^{(2)}(\mathbf{x}) p_t(\mathbf{x}) - f_t^{(2)}(\mathbf{x})\partial^{(2)} p_t(\mathbf{x}) \tag{1.42}$$

$$- \partial^{(3)} f_t^{(3)}(\mathbf{x}) p_t(\mathbf{x}) - f_t^{(3)}(\mathbf{x})\partial^{(3)} p_t(\mathbf{x})\Bigg) \tag{1.43}$$

$$= \mathrm{d}t\nu_1\nu_2\nu_3\left(-\nabla\cdot\left(f_t(\mathbf{x})p_t(\mathbf{x})\right)\right). \tag{1.44}$$

where ∇ is the gradient operator, $\nabla = [\partial^{(1)}, \partial^{(2)}, \partial^{(3)}]^\intercal$.

Brownian Motion Contribution To sum up the total increase in probability mass inside the volume due to Brownian motion, we will consider the distribution over jumps that occur due to the Brownian motion in time step $\mathrm{d}t$. We will consider jumps that take the particle from position $\mathbf{x}$ to position $\mathbf{x}+\boldsymbol{\mu}$ for $\boldsymbol{\mu}\in\mathbb{R}^3$. We let $\mathbb{P}(\boldsymbol{\mu}|\mathbf{x})$ be the jump distribution for time step $\mathrm{d}t$. Using our construction of Brownian motion, we have

$$\mathbb{P}(\boldsymbol{\mu}|\mathbf{x}) = \prod_{d=1}^{3}\left(\frac{1}{2}\delta(\mu_d - g_t\sqrt{\mathrm{d}t}) + \frac{1}{2}\delta(\mu_d + g_t\sqrt{\mathrm{d}t})\right). \tag{1.45}$$

We will now consider a system with Brownian motion but without drift. We let $\tilde{p}_t$ be the probability density at time t for this Brownian motion only system. At location $\mathbf{x}$, we can relate the probability density at time $t + \mathrm{d}t$ to an integral over the probability density at time t by adding up all the possible ways a particle could jump to location $\mathbf{x}$ in time step $\mathrm{d}t$,

$$\tilde{p}_{t+\mathrm{d}t}(\mathbf{x}) = \int \tilde{p}_t(\mathbf{x} - \boldsymbol{\mu})\mathbb{P}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu}. \tag{1.46}$$

Our first step is to expand $\tilde{p}_t(\mathbf{x} - \boldsymbol{\mu})$ using a Taylor expansion,

$$\tilde{p}_t(\mathbf{x} - \boldsymbol{\mu}) = \tilde{p}_t(\mathbf{x}) - \sum_{d=1}^{3} \partial^{(d)}\tilde{p}_t(\mathbf{x})\mu_d + \frac{1}{2}\sum_{i=1}^{3}\sum_{k=1}^{3} \mu_i \frac{\partial^2 \tilde{p}_t(\mathbf{x})}{\partial x_i \partial x_k}\mu_k + \text{higher order terms}. \tag{1.47}$$

Integrating the Taylor expansion, ignoring higher order terms, gives

$$\begin{aligned}
\tilde{p}_{t+\mathrm{d}t}(\mathbf{x}) &= \int \left(\tilde{p}_t(\mathbf{x}) - \sum_d \partial^{(d)}\tilde{p}_t(\mathbf{x})\mu_d + \frac{1}{2}\sum_i\sum_k \mu_i \frac{\partial^2 \tilde{p}_t(\mathbf{x})}{\partial x_i \partial x_k}\mu_k\right)\mathbb{P}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} && (1.48)\\
&= \tilde{p}_t(\mathbf{x}) - \sum_d \partial^{(d)}\tilde{p}_t(\mathbf{x}) \int \mu_d \mathbb{P}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} + && (1.49)\\
&\quad \frac{1}{2}\sum_i\sum_k \frac{\partial^2 \tilde{p}_t(\mathbf{x})}{\partial x_i \partial x_k} \int \mu_i\mu_k \mathbb{P}(\boldsymbol{\mu}|\mathbf{x})\mathrm{d}\boldsymbol{\mu} && (1.50)\\
&= \tilde{p}_t(\mathbf{x}) - \sum_d \partial^{(d)}\tilde{p}_t(\mathbf{x}) \left(\frac{1}{2}g_t\sqrt{\mathrm{d}t} - \frac{1}{2}g_t\sqrt{\mathrm{d}t}\right) + && (1.51)\\
&\quad \frac{1}{2}\sum_i\sum_k \frac{\partial^2 \tilde{p}_t(\mathbf{x})}{\partial x_i \partial x_k} \left(\delta\{i \neq k\}\left[\frac{1}{2}g_t^2\mathrm{d}t - \frac{1}{2}g_t^2\mathrm{d}t\right] + \delta\{i = k\}\left[g_t^2\mathrm{d}t\right]\right) && (1.52)\\
&= \tilde{p}_t(\mathbf{x}) + \frac{1}{2}\sum_i \frac{\partial^2 \tilde{p}_t(\mathbf{x})}{\partial^2 x_i} g_t^2 \mathrm{d}t && (1.53)\\
&= \tilde{p}_t(\mathbf{x}) + \frac{1}{2}\mathrm{d}t g_t^2 \Delta \tilde{p}_t(\mathbf{x}). && (1.54)
\end{aligned}$$

where Δ is the Laplacian operator $\Delta = \nabla \cdot \nabla$.

Combining Contributions We can now consider the total change in probability mass inside the infintesimal volume due to both the drift and the Brownian motion. We relate this to the time derivative of the probability mass. We have

$$\begin{aligned}
p_{t+\mathrm{d}t}(\mathbf{x})\nu_1\nu_2\nu_3 &= \nu_1\nu_2\nu_3\left(p_t(\mathbf{x}) + \mathrm{d}t\left(-\nabla \cdot (f_t(\mathbf{x})p_t(\mathbf{x}))\right) + \frac{1}{2}\mathrm{d}t g_t^2 \Delta p_t(\mathbf{x})\right) && (1.55)\\
&= \nu_1\nu_2\nu_3\left(p_t(\mathbf{x}) + \partial_t p_t(\mathbf{x})\mathrm{d}t + \text{higher order terms}\right), && (1.56)
\end{aligned}$$

where the second line is simply a Taylor expansion of $p_{t+\mathrm{d}t}(\mathbf{x})$ in the time variable. Ignoring higher order terms, we therefore find that

$$\partial_t p_t(\mathbf{x}) = -\nabla \cdot (f_t(\mathbf{x}) p_t(\mathbf{x})) + \frac{1}{2} g_t^2 \Delta p_t(\mathbf{x}), \tag{1.57}$$

giving us the Fokker-Planck-Kolmogorov equation in continuous state spaces.

FPK equation for CTMCs

The FPK equation for CTMCs can also be derived as an accounting of incoming and outgoing probability mass. If we consider a current state $\mathbf{x}_t$, then the total incoming probability mass in infinitesimal timestep $\mathrm{d}t$ will be $\sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t) p_t(j) \mathrm{d}t$. The total outgoing probability mass will be $\sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j) p_t(\mathbf{x}_t) \mathrm{d}t$. The difference between these two gives the overall accumulation of probability mass with state $\mathbf{x}_t$,

$$\partial_t p_t(\mathbf{x}_t) \mathrm{d}t = \sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t) p_t(j) \mathrm{d}t - \sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j) p_t(\mathbf{x}_t) \mathrm{d}t. \tag{1.58}$$

Dividing by $\mathrm{d}t$ provides the FPK equation for CTMCs,

$$\partial_t p_t(\mathbf{x}_t) = \sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t) p_t(j) - \sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j) p_t(\mathbf{x}_t). \tag{1.59}$$

In Figure 1.5, we pictorially represent the incoming and outgoing flows leading to the FPK equation.

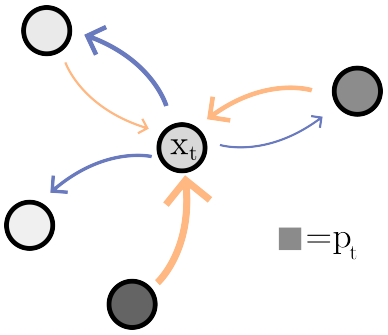


Figure 1.5: The accumulation of probability mass in state $\mathbf{x}_t$ is the difference between incoming and outgoing flows, $\partial_t p_t(\mathbf{x}_t) = \sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t) p_t(j) - \sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j) p_t(\mathbf{x}_t)$. Incoming flows are depicted in orange whereas outgoing flows are depicted in blue. The probability mass within each state p_t is depicted as the shade of grey. The width of the arrow represents the overall amount of flow between a pair of states, $R_t(j, \mathbf{x}_t) p_t(j)$.

If we make the definition that the diagnoal entries of our rate matrix are equal to the negative row sums i.e. $R_t(\mathbf{x}_t, \mathbf{x}_t) = -\sum_{j \neq \mathbf{x}_t)} R_t(\mathbf{x}_t, j)$ then the FPK equation can be written in a concise form as

$$\partial_t p_t(\mathbf{x}_t) = \sum_j R_t(\mathbf{x}_t, j) p_t(j). \tag{1.60}$$

1.7 Diffusion and Flow-Based Models

Using the tools of stochastic process theory, we are now able to describe the construction of generative processes, specifically diffusion and flow-based models on both continuous and discrete data spaces. Both techniques use a prior imparted structure to define a desired set of marginal distributions $\{p_t(\mathbf{x}_t)\}_{t=0}^{t=1}$ through time t. The initial marginal distribution at time $t = 0$, $p_0(\mathbf{x}_0)$, is a simple distribution that is easy to sample whilst the final distribution $p_1(\mathbf{x}_1)$ is equal to the data distribution $p_{\text{data}}(\mathbf{x}_1)$. The aim is to learn a process that has p_t as its marginal distribution at all times t such that simulating this process from $t = 0$ to $t = 1$ generates data from noise.

1.7.1 Diffusion Models

Diffusion models (Sohl-Dickstein et al. 2015; Ho et al. 2020; Song et al. 2021b) impart structure onto the generative process by first defining a corruption process through which data samples are transformed into pure noise. This corruption process defines a series of marginal distributions. The generative model then aims to learn an SDE that has those same marginal distributions but can be simulated in the opposite temporal direction, from noise to data.

Continuous Space Diffusion Models We begin by defining a corruption SDE with drift $f_t(\mathbf{x})$ and diffusion coefficient g_t that runs from time $t = 1$ to $t = 0$. The drift and diffusion coefficient are chosen so that the information in $p_1(\mathbf{x}_1) = p_{\text{data}}(\mathbf{x}_1)$ is gradually destroyed over time and the process should converge to the simple

$p_0(\mathbf{x}_0)$ at time $t = 0$. The FPK equation allows us to relate this defined SDE to the progression of marginals,

$$-\partial_t p_t(\mathbf{x}_t) = -\nabla \cdot (f_t(\mathbf{x}_t) p_t(\mathbf{x}_t)) + \frac{1}{2} g_t^2 \Delta p_t(\mathbf{x}_t), \tag{1.61}$$

where we note the negative sign on the LHS of Eq. 1.61 is due to this SDE corresponding to the reverse time progression of marginals from time $t = 1$ to time $t = 0$. Now, suppose we wish to find some drift $\tilde{f}_t(\mathbf{x}_t)$ and diffusion coefficient $\tilde{g}_t$ of an SDE such that simulating with this SDE forward in time from $t = 0$ to $t = 1$ results in the same marginal distributions as defined by Eq. 1.61. In other words, we wish to satisfy

$$\partial_t p_t(\mathbf{x}_t) = -\nabla \cdot \left(\tilde{f}_t(\mathbf{x}_t) p_t(\mathbf{x}_t)\right) + \frac{1}{2} \tilde{g}_t^2 \Delta p_t(\mathbf{x}_t). \tag{1.62}$$

Note the lack of a negative sign this time on the LHS of Eq. 1.62 as our SDE corresponds to the forward in time progression from $t = 0$ to $t = 1$. Since both of these equations correspond to the same marginal progression, we can equate them

$$\nabla \cdot \left(\tilde{f}_t(\mathbf{x}_t) p_t(\mathbf{x}_t)\right) - \frac{1}{2} \tilde{g}_t^2 \Delta p_t(\mathbf{x}_t) = -\nabla \cdot (f_t(\mathbf{x}_t) p_t(\mathbf{x}_t)) + \frac{1}{2} g_t^2 \Delta p_t(\mathbf{x}_t) \tag{1.63}$$

$$\nabla \cdot \left(\tilde{f}_t(\mathbf{x}_t) p_t(\mathbf{x}_t)\right) = -\nabla \cdot (f_t(\mathbf{x}_t) p_t(\mathbf{x}_t)) + \frac{1}{2} \left(g_t^2 + \tilde{g}_t^2\right) \Delta p_t(\mathbf{x}_t) \tag{1.64}$$

$$\nabla \cdot \left(\tilde{f}_t(\mathbf{x}_t) p_t(\mathbf{x}_t)\right) = -\nabla \cdot (f_t(\mathbf{x}_t) p_t(\mathbf{x}_t)) + \tag{1.65}$$

$$\frac{1}{2} \left(g_t^2 + \tilde{g}_t^2\right) \nabla \cdot (\nabla \log p_t(\mathbf{x}_t) p_t(\mathbf{x}_t)) \tag{1.66}$$

$$\nabla \cdot \left(\tilde{f}_t(\mathbf{x}_t) p_t(\mathbf{x}_t)\right) = \nabla \cdot \left(\left(-f_t(\mathbf{x}_t) + \frac{1}{2} \left(g_t^2 + \tilde{g}_t^2\right) \nabla \log p_t(\mathbf{x}_t)\right) p_t(\mathbf{x}_t)\right). \tag{1.67}$$

From this relation, we can infer that a suitable drift $\tilde{f}_t$ to achieve the same marginal progression is

$$\tilde{f}_t(\mathbf{x}_t) = -f_t(\mathbf{x}_t) + \frac{1}{2} \left(g_t^2 + \tilde{g}_t^2\right) \nabla \log p_t(\mathbf{x}_t). \tag{1.68}$$

We see that this is achievable for any value of $\tilde{g}_t \geq 0$. The choice of $\tilde{g}_t = g_t$ corresponds to the choice made in Song et al. 2021b and corresponds to the celebrated reverse time diffusion equation (Anderson 1982). The choice of $\tilde{g}_t = 0$

corresponds to using a deterministic ODE to simulate the marginal progression and is referred to as the 'Probability Flow ODE' form of a diffusion model within the literature (Song et al. 2021b).

The implication of this result is that if we can calculate $\tilde{f}_t$, then we could simulate the SDE defined by $\tilde{f}_t$ and $\tilde{g}_t$ from $t = 0$ to $t = 1$ thus converting samples of noise to samples from the data distribution. The difficulty in calculating $\tilde{f}_t$ is in the calculation of the score, $\nabla \log p_t(\mathbf{x}_t)$, which is analytically intractable. To circumvent this, Song et al. 2019 apply the denoising score matching technique (Vincent 2011; Hyvärinen et al. 2005) to learn an approximation to the score, $s_\theta(\mathbf{x}_t, t)$, over all times t,

$$\min_\theta \quad \mathbb{E}_{\mathcal{U}(t;0,1)p_{\text{data}}(\mathbf{x}_1)p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)} \left[\left|\left| s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) \right|\right|^2 \right], \tag{1.69}$$

which has the desired optimum because the solution to the L2 regression problem is the conditional expectation of the target variable and $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) = \mathbb{E}_{p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)} \left[\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_1) \right]$. We make particular note of how the learning objective for our score neural network is an expectation over losses computed at individual intermediate points t in the the generative process. This avoids any need to simulate a process at training time because $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)$ is available in closed form. In addition, we do not need to backpropagate through an entire stack of probabilistic layers which led to training difficulties in prior multi-step generative models as discussed in Section 1.4. Furthermore, we describe in Section 1.7.4 how this objective controls the likelihood of the generative process leading to mode-covering behaviour of the resulting generative model. We note that our generative process defined by the drift in Eq. 1.68 operates in a fixed dimensional space $\mathbf{x}_t \in \mathbb{R}^d$, $\forall t$ and we explore ways to allow the process to generate across different dimensions in Chapter 4.

Discrete Space Diffusion Models These models employ CTMCs as the base stochastic process in order to generate discrete data, $\mathbf{x} \in \{1, \dots, S\}$. We defer discussion of the multi-dimensional case to Chapter 3. A corruption process that gradually destroys information in the data is again employed to define the set of marginals $\{p_t(\mathbf{x}_t)\}_{t=0}^{t=1}$. The corruption process is defined through a choice of rate matrix R_t and runs from time $t = 1$ back to time $t = 0$ as in the continuous space case.

One example choice is to corrupt data by causing transitions to other states chosen uniformly at random, $R_t = \beta_t \mathbf{1}\mathbf{1}^\intercal - \beta_t SI$, where $\mathbf{1}$ is the vector of all ones and β_t is a scalar that describes the speed at which these transitions happen. The β_t schedule is chosen such that after a unit interval of time, samples following this process are distributed uniformly in the state space. With this choice of corruption, our simple starting distribution is the uniform distribution over the state space, $p_0(\mathbf{x}_0) = \frac{1}{S}$. Other choices are explored in Chapter 3.

Our choice of corruption R_t sets the marginal progression through the FPK equation on discrete spaces,

$$-\partial_t p_t(\mathbf{x}_t) = \sum_j R_t(j, \mathbf{x}_t) p_t(j). \tag{1.70}$$

Now, we would like to find a new CTMC with rate matrix $\tilde{R}_t$ that goes through the same marginal progression, however, runs forward in time from $t = 0$ to $t = 1$. In other words, we desire

$$\partial_t p_t(\mathbf{x}_t) = \sum_j \tilde{R}_t(j, \mathbf{x}_t) p_t(j). \tag{1.71}$$

Equating 1.70 and 1.71, we obtain

$$\sum_j \tilde{R}_t(j, \mathbf{x}_t) p_t(j) = -\sum_j R_t(j, \mathbf{x}_t) p_t(j). \tag{1.72}$$

This is not straightforward to solve directly, but we can propose a solution and verify that it indeed satisfies Eq. 1.72. This proposed solution is

$$\tilde{R}_t(\mathbf{x}_t, j) = \begin{cases} R_t(j, \mathbf{x}_t)\frac{p_t(j)}{p_t(\mathbf{x}_t)} & \text{for } \mathbf{x}_t \neq j \\ -\sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t)\frac{p_t(j)}{p_t(\mathbf{x}_t)} & \text{for } \mathbf{x}_t = j. \end{cases} \tag{1.73}$$

To verify this solution indeed works, we substitute it into the LHS of Eq. 1.72,

$$\begin{aligned} \text{LHS} &= \sum_{j \neq \mathbf{x}_t} \tilde{R}_t(j, \mathbf{x}_t) p_t(j) + \tilde{R}_t(\mathbf{x}_t, \mathbf{x}_t) p_t(\mathbf{x}_t) && (1.74) \\ &= \sum_{j \neq \mathbf{x}_t} R_t(\mathbf{x}_t, j)\frac{p_t(\mathbf{x}_t)}{p_t(j)} p_t(j) - \sum_{j \neq \mathbf{x}_t} R_t(j, \mathbf{x}_t)\frac{p_t(j)}{p_t(\mathbf{x}_t)} p_t(\mathbf{x}_t) && (1.75) \\ &= -\sum_{j} R_t(j, \mathbf{x}_t) p_t(j) && (1.76) \\ &= \text{RHS}. && (1.77) \end{aligned}$$

We have therefore found a rate matrix, $\tilde{R}_t$ that we can use to simulate a CTMC and proceed through the marginal progression from noise at $t = 0$ towards the data distribution at time $t = 1$. In the same way that in the continuous space case, this required us to know the intractable score, $\nabla \log p_t(\mathbf{x}_t)$, here we instead need to know the intractable probability ratio $\frac{p_t(j)}{p_t(\mathbf{x}_t)}$. In Chapters 3 and 6 we explore methods to parameterize and learn this ratio. Furthermore, the $\tilde{R}_t$ chosen here corresponds to the time reversal, similar to the choice $\tilde{g}_t = g_t$ for continuous diffusion models. Other choices also give the same marginal progression as we again explore in Chapters 3 and 6.

1.7.2 Flow-Based Models

In contrast to diffusion model's utilization of a corruption process to define the marginal progression, $\{p_t(\mathbf{x}_t)\}_{t=0}^{t=1}$, flow-based models (Peluchetti 2021; Liu et al. 2022; Albergo et al. 2023b; Lipman et al. 2022) instead use conditional processes, $\{p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\}_{t=0}^{t=1}$, that converge to a given datapoint $\mathbf{x}_1$. These conditional processes are mixed together using the data distribution to give the final unconditional generative process marginals, $p_t(\mathbf{x}_t) = \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right]$. We require the conditional distribution to be simple and not dependent on $\mathbf{x}_1$ at $t = 0$, $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = p_{\text{simple}}(\mathbf{x}_0)$. This results in a simple unconditional distribution too,

$p_0(\mathbf{x}_0) = \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[p_{\text{simple}}(\mathbf{x}_0)\right] = p_{\text{simple}}(\mathbf{x}_0)$. We require the conditional distribution to be a delta mass on $\mathbf{x}_1$ at time $t = 1$, $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = \delta(\mathbf{x}_t - \mathbf{x}_1)$. This sets the marginal at $t = 1$ to the data, $p_1(\mathbf{x}_1) = \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[\delta(\mathbf{x}_t - \mathbf{x}_1)\right] = p_{\text{data}}(\mathbf{x}_1)$. This means our $\{p_t(\mathbf{x}_t)\}_{t=0}^{t=1}$, marginals transition from p_{simple} towards p_{data} as desired.

Continuous Space Flow-Based Models For continuous data, the conditional marginal distributions are defined using Gaussian distributions. A simple choice meeting our requirements could be $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_t; t\mathbf{x}_1, (1-t)^2 I)$. Once we have defined $p_{t|1}$ we use the FPK equation to infer which conditional process achieves the desired conditional marginals,

$$\partial_t p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = -\nabla \cdot \left(v_t(\mathbf{x}_t|\mathbf{x}_1) p_t(\mathbf{x}_t|\mathbf{x}_1)\right). \tag{1.78}$$

Here we have assumed a noise free process to align with how flow-based models are commonly presented in the literature, however stochastic versions are possible (Peluchetti 2021; Albergo et al. 2023a). To continue with our simple running example, if $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_t; t\mathbf{x}_1, (1-t)^2 I)$ then $v_t(\mathbf{x}_t|\mathbf{x}_1) = \frac{\mathbf{x}_t - \mathbf{x}_1}{1-t}$ is one solution to Eq. 1.78. We now take the expectation with respect to p_{data} of both sides of Eq. 1.78,

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[\partial_t p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[-\nabla \cdot \left(v_t(\mathbf{x}_t|\mathbf{x}_1) p_t(\mathbf{x}_t|\mathbf{x}_1)\right)\right] \tag{1.79}$$

$$\partial_t \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right] = -\nabla \cdot \left(\mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[v_t(\mathbf{x}_t|\mathbf{x}_1) p_t(\mathbf{x}_t|\mathbf{x}_1)\right]\right) \tag{1.80}$$

$$\partial_t p_t(\mathbf{x}_t) = -\nabla \cdot \left(\mathbb{E}_{p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)}\left[v_t(\mathbf{x}_t|\mathbf{x}_1)\right] p_t(\mathbf{x}_t)\right), \tag{1.81}$$

where $p_{1|t}(\mathbf{x}_1|\mathbf{x}_t) = \frac{p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) p_{\text{data}}(\mathbf{x}_1)}{p_t(\mathbf{x}_t)}$. Therefore, we have shown that if we have access to $v_t(\mathbf{x}_t) := \mathbb{E}_{p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)}\left[v_t(\mathbf{x}_t|\mathbf{x}_1)\right]$ then we can perform generative modelling by simulating a process with the unconditional marginal progression starting from pure noise and ending at the data distribution. As in the continuous space diffusion case, this guiding drift term is intractable to calculate analytically but can be learnt as the solution to an L2 regression problem (Peluchetti 2021; Liu et al. 2022; Albergo et al. 2023b; Lipman et al. 2022)

$$\min_{\theta} \quad \mathbb{E}_{\mathcal{U}(t;0,1) p_{\text{data}}(\mathbf{x}_1) p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)}\left[||v_\theta(\mathbf{x}_t, t) - v_t(\mathbf{x}_t|\mathbf{x}_1)||^2\right]. \tag{1.82}$$

Flows on continuous spaces can be extended to operate on manifold data by defining the flow to move across the manifold surface. We explore the application of these approaches to tasks in protein design in Chapter 5.

Discrete Space Flow-Based Models In the discrete data case, we can follow the exact same recipe as used for continuous data. A simple choice for our conditional marginals is $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = t\delta\{\mathbf{x}_t = \mathbf{x}_1\} + (1-t)\frac{1}{S}$ i.e. the conditional process linearly interpolates between a delta function on the conditioning datapoint and the uniform distribution on the state space. It is not trivial to obtain a conditional rate matrix $R_t(\mathbf{x}_t, j|\mathbf{x}_1)$ that corresponds to the conditional marginal distributions $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)$ and we describe in Chapter 6 how this can be achieved. Now, suppose we have access to a $R_t(\mathbf{x}_t, j|\mathbf{x}_1)$ that is consistent with our desired conditional marginals, i.e.

$$\partial_t p_{t|1}(\mathbf{x}_t|\mathbf{x}_1) = \sum_j R_t(j, \mathbf{x}_t|\mathbf{x}_1) p_t(j|\mathbf{x}_1). \tag{1.83}$$

We can apply the same trick as the continuous space case by taking the expectation of both sides of Eq. 1.83,

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[\partial_t p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right] = \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[\sum_j R_t(j, \mathbf{x}_t|\mathbf{x}_1) p_t(j|\mathbf{x}_t)\right] \tag{1.84}$$

$$\partial_t \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right] = \sum_j \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[R_t(j, \mathbf{x}_t|\mathbf{x}_1) p_t(j|\mathbf{x}_1)\right] \tag{1.85}$$

$$\partial_t p_t(\mathbf{x}_t) = \sum_j \mathbb{E}_{p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)}\left[R_t(j, \mathbf{x}_t|\mathbf{x}_1)\right] p_t(j). \tag{1.86}$$

We recognise the final line as an unconditional FPK equation with rate matrix given by,

$$R_t(\mathbf{x}_t, j) := \mathbb{E}_{p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)}\left[R_t(j, \mathbf{x}_t|\mathbf{x}_1)\right]. \tag{1.87}$$

This rate matrix is intractable to calculate directly, we cover how it can be approximated with a neural network in Chapter 6. With this rate matrix in hand, we can simulate the unconditional marginals from noise to data and thus perform generative modelling.

1.7.3 Comparing Diffusions and Flows

With a unifying description of diffusion and flow-based models, we can here describe some of the key differences in these approaches. Diffusion models define the generative process through first constructing a corruption process which is defined via process infinitesimal quantities: the drift and diffusion coefficient in the continuous case and the rate matrix in the discrete case. These are then 'integrated' through the FPK equation to result in the final set of marginals $\{p_t\}_{t=0}^{t=1}$ used for the generative process. In contrast, for flow-based models, the generative process is constructed through first defining conditional marginal distributions. These are then 'differentiated' via the FPK equation to find the corresponding conditional process infinitesimals: the velocity and the conditional rate matrix.

In many cases, these two approaches can yield the same generative process, as we briefly touch on in Chapter 5 and is shown by Albergo et al. 2023b; Zheng et al. 2023. However, operating at the distributional level first (as in flow-based models) can have advantages in some cases. For example, in the discrete case, integrating the corruption rate matrix to find the required $p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)$ conditional marginals for a diffusion model is often a difficult operation to do involving the matrix exponential as described in Chapter 3. Instead, starting with conditional marginals and differentiating down to the rate matrix as in Chapter 6 is an easier calculation to carry out. Treating the marginal distributions as the first class citizens as opposed to process infinitesimals also allows the user to reason about the corruption schedule versus t in a more natural way and makes clearer the range of process dynamics that are available for a given marginal progression.

1.7.4 Training Objectives and Likelihoods

In the previous section we discussed objectives that can be used to train networks that parameterize our generative processes. Through simulation, these networks in turn define distributions over the data space and therefore it is important to understand how the training objective links to the likelihood assigned by the

generative process to the training datapoints. Specifically, does minimizing the objective correspond to maximizing the log-likelihood of the model on the training data?

We first consider the continuous state space diffusion model case. It was shown by Song et al. 2021a; Huang et al. 2021 that a weighted form of the denoising score matching loss in Eq. 1.69 is equal to an upper bound on the negative log-likelihood up to a constant. Reproducing the result here, we have

$$-\mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)}\left[\log p_\theta^{\text{SDE}}(\mathbf{x}_1)\right] \leq \tag{1.88}$$

$$\frac{1}{2}\int_0^1 \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)}\left[g_t^2 \left|\left|s_\theta(\mathbf{x}_t,t) - \nabla_{\mathbf{x}_t}\log p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)\right|\right|_2^2\right]\mathrm{d}t + \text{const}, \tag{1.89}$$

where $p_\theta^{\text{SDE}}(\mathbf{x}_1)$ is the distribution over final samples when simulating the generative SDE,

$$\mathrm{d}\mathbf{x} = \left(-f_t(\mathbf{x}_t) + g_t^2 s_\theta(\mathbf{x}_t,t)\right)\mathrm{d}t + g_t \mathrm{d}w_t, \tag{1.90}$$

from $t = 0$ to $t = 1$. We can therefore see that achieving a low denoising score matching loss is directly related to improving the log-likelihood of the generative model defined by the generative process and leads to desirable mode-covering behaviour. Relating back to our prior discussions on 'inversions' within generative models, this log-likelihood result is derived by utilizing the corruption process as that inverting tool. The corruption process describes the 'inputs' to the model (corrupted samples) that correspond to 'outputs' of the model (slightly less corrupted samples).

When the generative process is a flow model that is simulated with an ODE, then controlling an L2 distance on the drift is not sufficient to control the likelihood in general, as shown in Albergo et al. 2023a. Of course, the likelihood of the output of the model can be calculated using tools from Continuous Normalizing Flows (Chen et al. 2018a) however, the loss itself doesn't directly control this likelihood. Wildberger et al. 2023 refine this analysis and find that under sufficiently strong regularity conditions on the true score and vector field, the L2 loss can indeed

control the likelihood of the model.

In discrete state spaces, for diffusion style models, the objective used to learn the generative process is directly derived from a bound on the log-likelihood and is detailed in Chapter 3. For flow-based models where the required generative rate matrix is written directly in terms of a an expectation of a conditional rate with respect to a denoising distribution, Eq. 1.87, one can directly learn the $p_{1|t}(\mathbf{x}_1|\mathbf{x}_t)$ distribution using a cross entropy loss

$$\max_{\theta} \quad \mathbb{E}_{p_{\text{data}}(\mathbf{x}_1)p_{1|t}(\mathbf{x}_t|\mathbf{x}_1)} \left[\log p^{\theta}_{1|t}(\mathbf{x}_1|\mathbf{x}_t)\right]. \tag{1.91}$$

However, as we detail in Chapter 6, this does not directly control the log-likelihood as the diffusion style objective does.

1.8 Thesis Outline

In the remainder of this thesis, we will outline the extension of the generative process approach to generic data by expanding the state-spaces over which these processes can operate.

- Chapter 2 reviews the development of diffusion and flow-based approaches for continuous data and discrete data. We then detail existing methods for generic data modelling by first covering more classical approaches that utilize VAEs and GANs before moving onto other generative models including normalizing flows, autoregressive models and diffusion/flow-based models.

- Chapter 3 translates continuous space continuous time diffusion models to the discrete state space case through utilizing CTMCs. We derive a training objective, sampling procedure and design corruption processes that enable efficient training.

- Chapter 4 expands continuous space diffusion models to the case where the dimensionality of the data can change during generation. We construct a generative process with jumps that can add dimensions enabling the model

to decide on a suitable dimensionality for the generated sample based on any conditioning information.

- Chapter 5 utilizes a flow-based framework on manifold valued data for the task of protein motif-scaffolding. The flow on the space of rigid body motions generates a plausible scaffolding structure that can hold in place a given motif.

- Chapter 6 builds on continuous flow ideas by expanding the methodology to discrete data and multi-modal data. Through considering mixtures of conditional CTMCs, a generative CTMC can be constructed. We then combine a CTMC with a continuous flow creating a model capable of generating multi-modal data. We apply the model to protein co-design where a single model generates a protein's structure and sequence simultaneously.

- Chapter 7 concludes the thesis by summarizing the main arguments and discussing limitations and avenues for further work touching on extensions to highly multi-modal data and generative modelling fine-tuning.

1.9 Papers Omitted from Thesis

For completeness, this section lists published papers I have contributed to during the DPhil but are omitted from this thesis due to scope and space constraints.

1. Online Variational Filtering and Parameter Learning, ***A. Campbell****, *Y. Shi**, *T. Rainforth, A. Doucet*, NeurIPS 2021

2. Diffusion Schrödinger bridge matching, *Y. Shi, V. De Bortoli,* ***A. Campbell****, A. Doucet*, NeurIPS 2023

3. Fast Protein Backbone Generation with SE(3) Flow Matching, *J. Yim,* ***A. Campbell****, A. Y K Foong, M. Gastegger, J. Jiménez-Luna, S. Lewis, V. Garcia Satorras, B. S Veeling, R. Barzilay, T. Jaakkola, F. Noé*, Machine Learning in Structural Biology Workshop NeuriIPS 2023

4. Score-Optimal Diffusion Schedules, *C. Williams,* ***A. Campbell****, A. Doucet, S. Syed*, NeurIPS 2024

2

Literature Review

Contents

In this chapter we first review significant pieces of work in the development of diffusion and flow-based modelling. In contrast to the previous chapter which aimed to take a unifying perspective, this section discusses model parameterizations as they appeared in the original works. In the second half of this chapter, we review existing approaches for modelling generic data utilizing a range of generative model types from VAEs to autoregressive models.

2.1 Continuous Space Diffusion Models

The idea of generative models based on diffusion processes was first introduced in Sohl-Dickstein et al. 2015 who introduced the idea of a hierarchical VAE style model where the multi-level posterior is defined by a fixed and pre-defined noising process

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \tag{2.1}$$

where $\mathbf{x}_0$ is the datapoint and $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{x}_{t-1}\sqrt{1-\beta_t}, I\beta_t)$ is a Gaussian distribution that adds a small amount of noise to the previous state. For the decoder, a Markov chain is again used,

$$p_\theta(\mathbf{x}_{0:T-1}|\mathbf{x}_T) = \prod_{t=0}^{T-1} p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}), \tag{2.2}$$

with $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}) = \mathcal{N}(\mathbf{x}_t; \mu_\theta(\mathbf{x}_{t+1}, t+1), \Sigma_\theta(\mathbf{x}_{t+1}, t+1))$. The log-likelihood lower bound objective for the hierarchical VAE is then manipulated so that the entire probabilistic stack need not be evaluated to train the generative decoder but can be trained considering only a single layer at a time allowing for efficient training. Specifically, the objective is,

$$\mathcal{L} = -\sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)} \left[\mathrm{KL}\left(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\right)\right] + \tag{2.3}$$

$$\mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_1)} \left[\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] + \text{const.} \tag{2.4}$$

This is significant as it allows training a very deep hierarchical VAE a single layer at a time without needing to propagate gradients through a deep encoder stack. The model was parameterized with a simple convolutional architecture giving samples that did not look realistic.

Considering the stack of generative transition densities $\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ as a learned Markov chain, other work also considered the use of Markov chains to generate data. Song et al. 2019 aimed to generate data by using annealed Langevin dynamics. The Langevin dynamics update is of the form

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \frac{\alpha_i}{2}\nabla_{\mathbf{x}_{t-1}} \log p_{\sigma_i}(\mathbf{x}_{t-1}) + \sqrt{\alpha_i}\epsilon_t, \tag{2.5}$$

where $\epsilon_t \sim \mathcal{N}(\epsilon_t; 0, I)$, α_i is the step size and p_{σ_i} is the target distribution defined by noise level σ_i, $p_{\sigma_i}(\mathbf{x}) = \int p_{\text{data}}(\tilde{\mathbf{x}})\mathcal{N}(\mathbf{x}; \tilde{\mathbf{x}}, \sigma_i^2 I)$. The intractable score, $\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$ is approximated using a neural network, trained through denoising score matching

$$\min \quad ||s_\theta(\tilde{\mathbf{x}}, \sigma) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})||^2, \tag{2.6}$$

where $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 I)$ is the noising distribution used to define p_σ. The score network was parameterized with a modern U-Net architecture (Ronneberger et al. 2015) which then enabled high quality samples to be generated on small image datasets.

In 2020, two papers then took the initial ideas presented in this initial work and greatly popularized diffusion models as a viable generative modelling paradigm. The first by Ho et al. 2020 built upon the probabilistic framework of Sohl-Dickstein et al. 2015. They fixed the β_t hyperparameters, to a fixed schedule that linearly interpolates between 10^{-4} and 0.02. Refinements were also made to the parameterization and learning of the $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ generative transition kernel. If we consider the t point in the hierarchical model, by marginalizing the forward transition operators we find that $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon$ where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$ and $\epsilon \sim \mathcal{N}(\epsilon; 0, I)$. Ho et al. 2020 then define $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ through a noise predictor model $\epsilon_\theta(\mathbf{x}_t, t)$ that approximates the noise component ϵ in $\mathbf{x}_t$. The noise predictor is trained with an L2 loss over all noise levels,

$$\min_\theta \quad \mathbb{E}_{\mathcal{U}(t;1,T)p_{\text{data}}(\mathbf{x}_0)q(\mathbf{x}_t|\mathbf{x}_0)} \left[||\epsilon_\theta(\mathbf{x}_t, t) - \epsilon||^2\right] \tag{2.7}$$

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon, \quad \epsilon \sim \mathcal{N}(\epsilon; 0, I). \tag{2.8}$$

This loss can be derived by manipulating the terms $\text{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$, however, a weighting with respect to t appears inside the expectation. A key contribution of Ho et al. 2020 was to remove this weighting and train on an unweighted L2 loss which, in combination with a modern U-Net architecture, lead to high quality samples on high resolution image data.

The second paper released in 2020 that defined our current understanding of diffusion models is the work by Song et al. 2021b. Here, diffusion models were presented using the language of stochastic differential equations (SDEs) for the first time. The paper presented the ideas described in Section 1.7.1 in continuous state spaces. The previous view of diffusion models as a hierarchical VAE with a fixed posterior presented by Ho et al. 2020; Sohl-Dickstein et al. 2015 could then be understood as a certain discretization of an underlying SDE.

Since these initial seminal papers, notable further works have refined and extended the diffusion modelling paradigm in continuous spaces. To highlight key examples, Karras et al. 2022 analyzed the sampling and training dynamics of diffusion SDEs to create a high performance framework. They modified the schedule of noise as to reduce curvature in the generative process enabling fast and accurate simulation. They also derived loss weightings and network preconditioning factors so that all neural network training targets are zero mean and unit variance which conforms to well known rules of thumb for network training.

In a separate line of work, effort has been placed into improving the sampling efficiency of diffusion models which naively require T function evaluations to simulate the entire generative process. Luhman et al. 2021 exploit the deterministic sampling provided by the probability flow ODE (Song et al. 2021b) to define a mapping between prior samples $\mathbf{x}_T$ and synthetic datapoints $\mathbf{x}_0$. They then use a distillation loss to train a student network to directly output the synthetic datapoint $\mathbf{x}_0$ corresponding to the starting point $\mathbf{x}_T$. This idea is refined by Salimans et al. 2022 where instead of tasking the student network to directly learn the noise to data map from scratch, the student network is trained such that simulating a generative process defined by the student network with a coarse discretization schedule matches the generation trajectory of a teacher diffusion model sampled with a fine discretization schedule. This procedure is iterated setting the teacher model to be the student from the previous round until a single step model is learned. This iterative procedure

improves sample quality. Current methods for fast sampling, aim to learn the student models directly through a trajectory matching loss (Song et al. 2023b; Song et al. 2023a) and with the integration of adversarial style losses (Sauer et al. 2025; Kim et al. 2023).

2.2 Discrete Space Diffusion Models

Discrete diffusion models utilize the same fundamental idea as continuous space diffusion models by defining a corruption process that can be used to derive a scalable learning objective to train a generative process. The corruption process however is defined to operate over representations of discrete data. In this section, we split the approaches into those that operate directly in the discrete space and those that instead embed discrete data into a continuous representation.

2.2.1 Discrete Space Approaches

Sohl-Dickstein et al. 2015 first introduced diffusion models directly in discrete space, where, instead of parameterizing $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ and $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ using Gaussian distributions, they are parameterized using Binomial distributions. Specifically,

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = (1-\beta_t)\delta\{\mathbf{x}_t = \mathbf{x}_{t-1}\} + \frac{1}{2}\beta_t, \tag{2.9}$$

$$p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}) = \delta\{\mathbf{x}_t = 0\}f_\theta(\mathbf{x}_{t+1}, t+1) + \delta\{\mathbf{x}_t = 1\}(1 - f_\theta(\mathbf{x}_{t+1}, t+1)), \tag{2.10}$$

where f_θ is parameterized using the neural network. This again creates a process that gradually corrupts data in the forward direction by replacing bits with samples of uniform random noise. The generative process then reverses this and generates data from random noise. The model was demonstrated only on toy binary data.

After the popularization of diffusion models in continuous space, effort was then made to continue the development of diffusion in discrete space. Hoogeboom et al. 2021b experiment with categorical state spaces with $K > 2$ categories,

switching the Binomial noising process of Sohl-Dickstein et al. 2015 for a Categorical noising process,

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = (1-\beta_t)\delta\{\mathbf{x}_t = \mathbf{x}_{t-1}\} + \frac{1}{K}\beta_t. \tag{2.11}$$

For the generative direction, $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ is parameterized by having the neural network output a distribution over the data point predicted to correspond to $\mathbf{x}_{t+1}$, $p_\theta(\mathbf{x}_0|\mathbf{x}_{t+1})$. This is then converted into a single step distribution through marginalization, $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1}) = \sum_{\mathbf{x}_0} p_\theta(\mathbf{x}_0|\mathbf{x}_{t+1})q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0)$ where $q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0)$ can be calculated exactly from our definition of the noising process. The sum over $\mathbf{x}_0$ is tractable to carry out in high dimensions because $p_\theta(\mathbf{x}_0|\mathbf{x}_{t+1})$ is defined to factorize over dimensions. Hoogeboom et al. 2021b carry out experiments on text and segmentation maps demonstrating that diffusion models on discrete space can scale to complex high dimensional tasks.

Austin et al. 2021 further explore the possible styles of corruption process on discrete data. In addition to the 'uniform' noise style of corruption used by Hoogeboom et al. 2021b, they explore an 'absorbing state' style process that has a small probability to transition to a mask token for each timestep,

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \delta\{\mathbf{x}_{t-1} \neq M\}\Big[(1-\beta_t)\delta\{\mathbf{x}_t = \mathbf{x}_{t-1}\} + \delta\{\mathbf{x}_t = M\}\beta_t\Big] + \tag{2.12}$$

$$\delta\{\mathbf{x}_{t-1} = M\}\delta\{\mathbf{x}_t = M\}, \tag{2.13}$$

where we can see that once a position has transitioned to the mask state, it will remain there for the remainder of the noising process. For discrete data that is ordered, a discretized Gaussian style corruption was introduced that biases states to transition to nearby states in the ordering. Finally, a corruption process that biases tokens to transition to other token values that are similar in an embedding space was proposed for text data. Austin et al. 2021 found that the best performing corruption for text remained the absorbing state style. This may be expected due to absorbing state corruption resembling the generation process of autoregressive large language models that gradually generate text by revealing one token at a time. For

all of these corruption processes, the generative process can be parameterized using the definition of $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ through marginalization over $\mathbf{x}_0$ prediction as before.

2.2.2 Continuous Embedding Approaches

These styles of approach utilize standard continuous space diffusion model techniques to model discrete data by embedding the discrete data into a continuous space. This can be as simple as assigning a binary code to all discrete categories and then treating the $\{0, 1\}^d$ vector as real valued (Chen et al. 2022). The diffusion model learns that all data values are either 0 or 1 meaning that even though intermediate values during the diffusion process are not valid discrete datapoints, the final generated values are close to 0 or 1 which are easily converted back to discrete data through quantization.

This approach does not make use of any structure available when operating in continuous space. Dieleman et al. 2022 model text by embedding each discrete category in the vocabulary into a continuous vector which are jointly learned with the diffusion score model. The embedding learning signal is backpropagated from the denoising loss requiring regularization of the embeddings to avoid degenerate behaviour. Gulrajani et al. 2024 refine the approach by training a joint embedding + diffusion model for text generation using a well-posed log-likelihood bound meaning that embedding regularization is no longer required.

2.3 Flow-Based Models

As described in Section 1.7.2, flow-based models define a generative process through a mixture of conditional processes. These ideas were first explored by Peluchetti 2021 and then later expanded on by Liu et al. 2022; Albergo et al. 2023b; Lipman et al. 2022. In our initial exposition of flow-based ideas, we considered processes conditional only on $\mathbf{x}_1$, however, Peluchetti 2021 treat these models in their full generality, pinned on both $\mathbf{x}_0$ and $\mathbf{x}_1$ and in the stochastic case. Peluchetti 2021 starts by considering a simple SDE of the form,

$$\mathrm{d}\mathbf{x}_t = f_t(\mathbf{x}_t)\mathrm{d}t + g_t\mathrm{d}w_t, \tag{2.14}$$

which is initialized at $\mathbf{x}_0$. Using Doob's h-transform, this SDE is then converted into a pinned SDE that is guaranteed to hit a given point $\mathbf{x}_T$ at time T by modifying the drift with a score forcing term,

$$\mathrm{d}\mathbf{x}_t = \Big(f_t(\mathbf{x}_t) + g_t^2 \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_T|\mathbf{x}_t)\Big)\,\mathrm{d}t + g_t \mathrm{d}w_t, \tag{2.15}$$

where $\log p(\mathbf{x}_T|\mathbf{x}_t)$ is the transition probability between times t and T implied by the SDE in Eq. 2.14. The SDE in Eq. 2.15 therefore begins at $\mathbf{x}_0$ and ends at $\mathbf{x}_T$. Now, the key is to consider the mixture of these pinned processes with respect to a joint distribution over $\mathbf{x}_0$ and $\mathbf{x}_T$, $\Pi(\mathbf{x}_0, \mathbf{x}_T)$. As an example, we could set $\Pi(\mathbf{x}_0, \mathbf{x}_T) = p_{\text{prior}}(\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_T)$ to obtain a case similar to standard diffusion and flow-based models, however, the joint need not be factorized and the $\mathbf{x}_0$ distribution doesn't have to be simple. The SDE in Eq. 2.15, when started at $\mathbf{x}_0$ and pinned to finish at $\mathbf{x}_T$, has marginal distributions $\{p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_T)\}_{t=0}^{t=T}$. The aim is to find an SDE that has marginal distributions $\pi_t(\mathbf{x}_t) = \mathbb{E}_{\Pi(\mathbf{x}_0,\mathbf{x}_T)}\left[p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_T)\right]$. Peluchetti 2021 prove the following SDE has the desired $\pi_t(\mathbf{x}_t)$ marginals,

$$\mathrm{d}\mathbf{x}_t = \Big(f_t(\mathbf{x}_t) + g_t^2 \mathbb{E}_{p(\mathbf{x}_T|\mathbf{x}_t)}\left[\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_T|\mathbf{x}_t)\right]\Big)\,\mathrm{d}t + g_t \mathrm{d}w_t, \tag{2.16}$$

where $p(\mathbf{x}_T|\mathbf{x}_t)$ is the posterior over $\mathbf{x}_T$ when $\mathbf{x}_t$ is generated by first sampling the $\mathbf{x}_0, \mathbf{x}_T$ joint distribution and then sampling $p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_T)$ i.e.

$$p(\mathbf{x}_T|\mathbf{x}_t) = \frac{\int_{\mathbf{x}_0} \Pi(\mathbf{x}_0, \mathbf{x}_T)p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_T)\mathrm{d}\mathbf{x}_0}{\pi_t(\mathbf{x}_t)}. \tag{2.17}$$

The term $\mathbb{E}_{p(\mathbf{x}_T|\mathbf{x}_t)}\left[\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_T|\mathbf{x}_t)\right]$ is intractable to calculate directly but can be approximated with a neural network trained using the following objective,

$$\mathbb{E}_{\mathcal{U}(t;0,T)\Pi(\mathbf{x}_0,\mathbf{x}_T)p(\mathbf{x}_t|\mathbf{x}_0,\mathbf{x}_T)}\left[||\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_T|\mathbf{x}_t) - s_\theta(\mathbf{x}_t, t)||_2^2\right]. \tag{2.18}$$

To consider the use of this framework specifically to the generative modelling case, we again consider our simple example $\Pi(\mathbf{x}_0, \mathbf{x}_T) = p_{\text{prior}}(\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_T)$. At time T we have,

$$\pi_T(\mathbf{x}_T) = \int_{\mathbf{x}_0, \tilde{\mathbf{x}}_T} \Pi(\mathbf{x}_0, \tilde{\mathbf{x}}_T)p(\mathbf{x}_T|\mathbf{x}_0, \tilde{\mathbf{x}}_T)\mathrm{d}\mathbf{x}_0\mathrm{d}\tilde{\mathbf{x}}_T \tag{2.19}$$

$$= \int_{\mathbf{x}_0, \tilde{\mathbf{x}}_T} p_{\text{prior}}(\mathbf{x}_0) p_{\text{data}}(\tilde{\mathbf{x}}_T) \delta\{\mathbf{x}_T - \tilde{\mathbf{x}}_T\} \mathrm{d}\mathbf{x}_0 \mathrm{d}\tilde{\mathbf{x}}_T \tag{2.20}$$

$$= p_{\text{data}}(\mathbf{x}_T). \tag{2.21}$$

Therefore, by simulating the SDE in Eq. 2.16 we can transform samples from the prior distribution to the data distribution, using a score network trained in a very similar way to denoising score matching. This style of model can in fact be shown to be equivalent to diffusion models in this case (Gao et al. 2024). Peluchetti 2021 demonstrated the model on toy data.

In 2022, 3 papers (Liu et al. 2022; Albergo et al. 2023b; Lipman et al. 2022) independently developed a similar mixture of processes approach to generative modelling and demonstrated this model class can scale to high dimensional image datasets. Rather than constructing the generative modelling framework through considering pinned SDEs, these works use conditional ODEs as the building blocks. Specifically, they first consider an ODE that will hit a terminal point $\mathbf{x}_1$ at time $t = 1$ when initialized from any starting point $\mathbf{x}_0$. The simplest example is the 'linear interpolant' conditional ODE,

$$\mathrm{d}\mathbf{x}_t = (\mathbf{x}_1 - \mathbf{x}_0)\mathrm{d}t. \tag{2.22}$$

We can see that the conditional ODE has both information regarding the starting point $\mathbf{x}_0$ and the end point $\mathbf{x}_1$. To perform generative modelling, a mixture of these conditional ODEs is considered, with respect to a joint distribution $\Pi(\mathbf{x}_0, \mathbf{x}_1)$. As before, if the marginal distributions of the process defined by Eq. 2.22 are denoted $p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1)$ then we can obtain an ODE with marginal distributions $\pi_t(\mathbf{x}_t) = \int_{\mathbf{x}_0, \mathbf{x}_1} \Pi(\mathbf{x}_0, \mathbf{x}_1) p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1) \mathrm{d}\mathbf{x}_0 \mathrm{d}\mathbf{x}_1$ by simulating the following ODE,

$$\mathrm{d}\mathbf{x}_t = \mathbb{E}_{p(\mathbf{x}_0, \mathbf{x}_1|\mathbf{x}_t)} \left[\mathbf{x}_1 - \mathbf{x}_0\right] \mathrm{d}t, \tag{2.23}$$

where $p(\mathbf{x}_0, \mathbf{x}_1|\mathbf{x}_t)$ is the posterior over $\mathbf{x}_0$ and $\mathbf{x}_1$ when $\mathbf{x}_t$ is sampled from $\Pi(\mathbf{x}_0, \mathbf{x}_1) p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1)$ i.e.

$$p(\mathbf{x}_0, \mathbf{x}_1|\mathbf{x}_t) = \frac{\Pi(\mathbf{x}_0, \mathbf{x}_1) p(\mathbf{x}_t|\mathbf{x}_0, \mathbf{x}_1)}{\pi_t(\mathbf{x}_t)}. \tag{2.24}$$

The expectation $\mathbb{E}_{p(\mathbf{x}_0,\mathbf{x}_1|\mathbf{x}_t)}[\mathbf{x}_1 - \mathbf{x}_0]$ can again be approximated with a neural network trained with an L2 loss,

$$\min_{\theta} \quad \mathbb{E}_{\mathcal{U}(t;0,1)\Pi(\mathbf{x}_0,\mathbf{x}_1)p(\mathbf{x}_t|\mathbf{x}_0,\mathbf{x}_1)}\left[||s_\theta(\mathbf{x}_t,t) - (\mathbf{x}_1 - \mathbf{x}_0)||_2^2\right]. \tag{2.25}$$

Albergo et al. 2023b; Lipman et al. 2022 used $\Pi(\mathbf{x}_0,\mathbf{x}_1) = p_{\text{prior}}(\mathbf{x}_0)p_{\text{data}}(\mathbf{x}_1)$ in order to use these flows for generative modelling and achieved high quality samples on image datasets. Liu et al. 2022 additionally considered a non-factorized joint distribution $\Pi(\mathbf{x}_0,\mathbf{x}_1)$ and use the approach to learn mappings between arbitrary distributions with a low transport cost. This is achieved by iterating the learning procedure in Eq. 2.25. They start with an independent coupling between two distributions $\Pi(\mathbf{x}_0,\mathbf{x}_1) = p_0(\mathbf{x}_0)p_1(\mathbf{x}_1)$ and learn a velocity for an ODE that transports between $p_0(\mathbf{x}_0)$ and $p_1(\mathbf{x}_1)$. The insight of Liu et al. 2022 is to then iterate this procedure by pushing $p_0(\mathbf{x}_0)$ forward through the learned ODE to obtain a new coupling between $p_0(\mathbf{x}_0)$ and $p_1(\mathbf{x}_1)$, this time one that is not factorized. They show that this procedure reduces the transport cost between the starting point and end point. This technique is demonstrated on high dimensional image data with the method able to provide meaningful translations between vastly different image datasets such as animals to humans faces.

Further work built on these ideas, such as Albergo et al. 2023a which provided a unifying overview of deterministic and stochastic mixture of bridges approaches. Shi et al. 2024 also built on the iterative procedure of Liu et al. 2022 in the stochastic case, creating an algorithm that can approximate the Schrödinger Bridge between high dimensional distributions.

2.4 Generic Data Generative Models

The previous generative frameworks are usually constructed assuming data lies in a simple euclidean space $\mathbf{x} \in \mathbb{R}^d$. As generative model methodology has evolved, there have been parallel developments in their application to non-standard datatypes such as discrete data, data lying on a complex manifold or multi-modal data.

2.4.1 Variational Autoencoders

Miao et al. 2016; Bowman et al. 2015 are two early examples of VAEs developed to generate discrete text data. This is achieved by using a decoding distribution $p_\theta(\mathbf{x}|\epsilon)$ suitable for discrete data. Miao et al. 2016 utilize a factorized distribution parameterized by a softmax on the neural network output $p_\theta(\mathbf{x}^{1:D}|\epsilon) = \prod_{d=1}^{D} p_\theta(\mathbf{x}^d|\epsilon)$ whereas Bowman et al. 2015 use an autoregressive decoder parameterized by a recurrent neural network $p_\theta(\mathbf{x}^{1:D}|\epsilon) = \prod_{d=1}^{D} p_\theta(\mathbf{x}^d|\mathbf{x}^{1:d-1}, \epsilon)$. The VAE framework introduces challenges ranging from low quality samples in the factorized case to a failure to use the latent encoding due to a powerful decoder in the autoregressive case.

To model data that lies on a non-euclidean manifold, Falorsi et al. 2018 use a VAE with a latent space that is homeomorphic to the assumed manifold that the data lies on. This requires modifications to the encoder $q_\phi(\epsilon|\mathbf{x})$ such that it defines a distribution on the appropriate manifold. Falorsi et al. 2018 experiment specifically with rotation data on SO(3) utilizing a reparameterized encoder network that first generates a noise sample around the origin using the isomorphism between $\mathbb{R}^3$ and the tangent space of SO(3) at the origin. The noise is then rotated by another output of the neural network allowing the encoder to parameterize a flexible distribution over SO(3). To complete the generative model, a decoder is used to map from the manifold valued latent space back to the observed data space.

In order to perform generative modelling on multi-modal discrete and continuous data, Ma et al. 2020 define a two-stage hierarchical VAE with a joint distribution $p(\epsilon^A)p_\psi(\epsilon^B|\epsilon^A)p_\theta(\mathbf{x}|\epsilon^B)$. Here $\mathbf{x}$ is multi-modal with the final $p_\theta(\mathbf{x}|\epsilon^B)$ decoder splitting over D different modalities as $p_\theta(\mathbf{x}|\epsilon^B) = \prod_{d=1}^{D} p_\theta(\mathbf{x}_d|\epsilon_d^B)$. The separated data modality decoders $p_\theta(\mathbf{x}_d|\epsilon_d^B)$ are trained individually with their respective encoders in a pre-training stage before being integrated into the multi-modal joint model. The $p_\psi(\epsilon^B|\epsilon^A)$ latent decoder enables the model to capture dependencies between modalities.

2.4.2 Generative Adversarial Networks

Discrete data presents unique challenges for the training of GANs due to the non-differentiability of the state-space. The computation of the gradient of the training objective for standard GANs involves differentiating through the output of the generator, $\nabla_\theta D_\phi^{\text{pre}}(G_\theta(\epsilon))$ see Eq. 1.19. To circumvent the non-differentiability issue when generating sequences of text tokens $Y_{1:T}$, Yu et al. 2017 make use of the REINFORCE method from reinforcement learning (Williams 1992). The generator is parameterized autoregressively, $G_\theta(Y_t|Y_{1:t-1})$, whilst the discriminator operates on entire sequences, $D_\phi(Y_{1:T})$. The challenge is to obtain the generator gradient for timestep t. A value function is used $Q(Y_{1:t})$ which approximates $\mathbb{E}_{G_\theta(Y_{t+1:T}|Y_{1:t})}\left[D_\phi(Y_{1:T})\right]$ leading to a gradient,

$$\nabla_\theta \mathbb{E}_{G_\theta(Y_t|Y_{1:t-1})}\left[Q(Y_{1:t})\right] = \mathbb{E}_{G_\theta(Y_t|Y_{1:t-1})}\left[\nabla_\theta \log G_\theta(Y_t|Y_{1:t-1}) Q(Y_{1:t})\right]. \tag{2.26}$$

Due to the high variance of REINFORCE style methods, the adversarial approach failed to scale to complex text generation tasks.

GANs can also be adapted to operate on manifold data. For example, Dey et al. 2020 utilize a rotation invariant convolutional network (Cohen et al. 2016) to define a discriminator on image data that has no preferred orientation. A rotationally equivariant generator then learns to generate data of all orientations.

2.4.3 Normalizing Flows

Normalizing flows have also been applied to the generic data modelling problem. For discrete data, Tran et al. 2019, modify normalizing flow layers to operate in the discrete state space. The layers define deterministic transformations between input and output discrete points and are Jacobian free as there are no spatial volumes to track changes in. The stack of layers can then be thought of as a learned re-labelling of the discrete base distribution to match the observed data. The base distribution parameters can be learned through backpropagating a maximum likelihood objective, however, the transformation parameters require

discrete gradient estimation techniques (Bengio et al. 2013; Maddison et al. 2017; Jang et al. 2017).

Mathieu et al. 2020 extend the continuous normalizing flow formulation onto manifold valued data for example data lying on a Torus or Sphere. The velocity field defining the ODE map is first translated onto a tangent space valued vector that defines a flow on the manifold surface. The change of variables formula is also adapted utilizing the structure of the space, taking into account the Riemannian metric when accounting for changes in probability volumes.

Finally, continuous normalizing flows have been applied to data involving discrete jumps embedded in a continuous space by Chen et al. 2021. The standard generative process is combined with a point process to model jumps requiring the continuous change-of-variable formula to also be adapted to additionally sum over these discrete events. We utilize similar techniques in Chapter 4 to include jumps within a continuous diffusion generative process.

2.4.4 Autoregressive Models

Autoregressive models are those that model a complex data distribution $p_{\text{data}}(\mathbf{x})$ by parameterizing the model's joint distribution as a series of autoregressive distributions, $p_\theta(\mathbf{x}^{1:D}) = \prod_{d=1}^{D} p_\theta(\mathbf{x}^d|\mathbf{x}^{1:d-1})$. In contrast to VAEs, GANs and diffusion models, autoregressive models are most commonly used for discrete data with continuous data modelling being the rarer use-case. For discrete data, each autoregressive distribution is a categorical distribution parameterized by a softmax non-linearity applied to the output of an unconstrained neural network.

To model continuous data with autoregressive models, two styles of approach have been used. The first is to translate the structure of the data into discrete tokens. For example Gebauer et al. 2019, generate molecules in 3D space requiring a generative model over continuous positions that should be invariant to rigid body

rotations of the molecule. This is achieved by having the model output the relative distance between the new atom and the previously placed atoms. These relative distances are binned to provide a discrete modelling problem for the autoregressive model. Hayes et al. 2024 carry the general idea further with the aim of modelling continuous protein structures using a discrete autoregressive model. In a pretraining stage, a VAE with a discrete latent space (Van Den Oord et al. 2017) is trained to generate discrete encodings of local 3D protein structures that are invariant to rigid body rotations. This way, a standard discrete autoregressive model can be trained on the discrete tokens, obviating the need to handle complex continuous structure with the autoregressive distributions.

The second approach is to use a continuous distribution as the $p_\theta(\mathbf{x}^d|\mathbf{x}^{1:d-1})$ autoregressive distribution. This could be as simple as a mixture of Gaussians (Uria et al. 2013; Theis et al. 2015) or as complex as an entire diffusion model ran for each dimension (Zhou et al. 2024). In order to model multi-modal data, the above approaches for continuous data can be intermixed with standard discrete token modelling resulting in a powerful joint model of mixed-type data.

An additional benefit of using autoregressive models for complex data structures is that data of varying lengths is handled naturally. Whereas diffusion models, VAEs and GANs typically require that the total dimensionality of the data to be generated is set at training time, autoregressive models can avoid this by being parameterized to also output a probability over a 'stop token' (Sutskever et al. 2014). During generation, the conditional $p_\theta(\mathbf{x}_d|\mathbf{x}_{1:d-1})$ distributions are iteratively sampled until the stop token is produced enabling the model to define a distribution over sequence lengths.

One drawback of using autoregressive models for multi-modal modelling is the necessity in defining a way to segment and order the data such that the series of autoregressive distributions can be defined. For data such as images,

there are multiple ways this ordering could be defined e.g. raster-scan ordering or zig-zag ordering (Chen et al. 2018b). The chosen ordering can then affect the performance of the model. To circumvent this, some works have aimed to learn a single model amortized over possible orderings (Yang 2019; Hoogeboom et al. 2021a; Pannatier et al. 2024).

2.4.5 Diffusion and Flow-Based Models

In the previous discussion, diffusion and flow-based models have been described for continuous and discrete data. There have been works extending these approaches to other complex data types.

Diffusion models have been adapted for data lying on a non-euclidean manifold by defining the corruption and generative stochastic processes to occur directly on the manifold (De Bortoli et al. 2022; Huang et al. 2022). The denoising score matching objective of Song et al. 2021b requires extra care to translate in this case as the $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)$ term is not always analytically tractable and needs to be approximated. Sampling occurs via a geodesic random walk, with the score providing an update direction in the tangent space which is translated into movement along the manifold through the exponential map.

Chen et al. 2024 simplify these approaches with a flow-based model for general geometries. Instead of requiring the definition of an appropriate stochastic process on the manifold and approximation of $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)$, the flow matching approach trains to match a conditional vector field $v_t(\mathbf{x}_t|\mathbf{x}_1)$ which can be found analytically on a wider class of manifolds than $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t|\mathbf{x}_0)$. A deterministic flow is used for sampling which does not require the definition of a stochastic process on the manifold.

In order to model multi-modal data, methods for applying diffusion models to continuous and discrete valued variables can be combined. This is a common

problem when generating new molecules that consist of both discrete atom types and continuous atom positions. In Hoogeboom et al. 2022, the authors embed the discrete atom types as one-hot vectors which are then treated as an additional continuous variable. Building on this work, Hua et al. 2023; Vignac et al. 2023; Peng et al. 2023 use a discrete state space diffusion process for the atom types and atom connectivity graphs and achieve superior performance. All these approaches faced the challenge of scheduling between the different modalities. The relative speed of corruption for atom types versus the atom positions defines an approximate ordering in which these modalities are generated, either atom types first or atom positions first. A common finding is that performance is better when atom positions are generated first.

3

A Continuous Time Framework for Discrete Denoising Models

A Continuous Time Framework for Discrete Denoising Models

Andrew Campbell[1] **Joe Benton**[1] **Valentin De Bortoli**[2]

Tom Rainforth[1] **George Deligiannidis**[1] **Arnaud Doucet**[1]

[1]Department of Statistics, University of Oxford, UK [2]CNRS ENS Ulm, Paris, France
{campbell, benton, rainforth, deligian, doucet}@stats.ox.ac.uk
valentin.debortoli@gmail.com

Abstract

We provide the first complete continuous time framework for denoising diffusion models of discrete data. This is achieved by formulating the forward noising process and corresponding reverse time generative process as Continuous Time Markov Chains (CTMCs). The model can be efficiently trained using a continuous time version of the ELBO. We simulate the high dimensional CTMC using techniques developed in chemical physics and exploit our continuous time framework to derive high performance samplers that we show can outperform discrete time methods for discrete data. The continuous time treatment also enables us to derive a novel theoretical result bounding the error between the generated sample distribution and the true data distribution.

1 Introduction

Diffusion/score-based/denoising models [1, 2, 3, 4] are a popular class of generative models that achieve state-of-the-art sample quality with good coverage of the data distribution [5] all whilst using a stable, non-adversarial, simple to implement training objective. The general framework is to define a forward noising process that takes in data and gradually corrupts it until the data distribution is transformed into a simple distribution that is easy to sample. The model then learns to reverse this process by learning the logarithmic gradient of the noised marginal distributions known as the score.

Most previous work on denoising models operates on a continuous state space. However, there are many problems for which the data we would like to model is discrete. This occurs, for example, in text, segmentation maps, categorical features, discrete latent spaces, and the direct 8-bit representation of images. Previous work has tried to realize the benefits of the denoising framework on discrete data problems, with promising initial results [6, 7, 8, 9, 10, 11, 12, 13].

All of these previous approaches train and sample the model in discrete *time*. Unfortunately, working in discrete time has notable drawbacks. It generally forces the user to pick a partition of the process at training time and the model only learns to denoise at these fixed time points. Due to the fixed partition, we are then limited to a simple ancestral sampling strategy. In continuous time, the model instead learns to denoise for any arbitrary time point in the process. This complete specification of the reverse process enables much greater flexibility in defining the reverse sampling scheme. For example, in continuous state spaces, continuous time samplers that greatly reduce the sampling time have been devised [14, 15, 16, 17] as well as ones that improve sample quality [4, 18]. The continuous time interpretation has also enabled the derivation of interesting theoretical properties such as error bounds [19] in continuous state spaces.

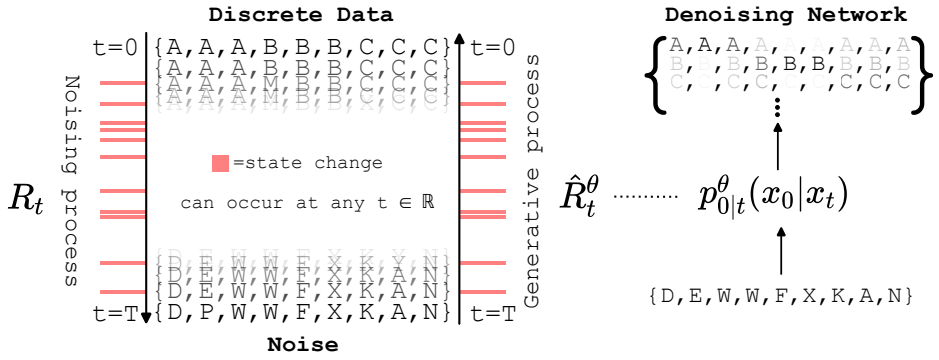


Figure 1: The forward noising process corrupts data according to R_t, the rate of corruption events at time t. The noising process' time reversal gives the generative process which is defined through $\hat{R}_t^\theta$, the rate of generative events at time t. $\hat{R}_t^\theta$ is parameterized through the denoising network, $p^\theta_{0|t}(x_0|x_t)$, which outputs categorical probabilities over clean x_0 values conditioned on a noisy x_t.

To allow these benefits to be exploited for discrete state spaces as well, we formulate a continuous time framework for discrete denoising models. Specifically, our contributions are as follows. We formulate the forward noising process as a Continuous Time Markov Chain (CTMC) and identify the generative CTMC that is the time-reversal of this process. We then bound the log likelihood of the generated data distribution, giving a continuous time equivalent of the ELBO that can be used for efficient training of a parametric approximation to the true generative reverse process. To efficiently simulate the parametric reverse process, we leverage tau-leaping [20] and propose a novel predictor-corrector type scheme that can be used to improve simulation accuracy. The continuous time framework allows us to derive a bound on the error between the true data distribution and the samples generated from the approximate reverse process simulated with tau-leaping. Finally, we demonstrate our proposed method on the generative modeling of images from the CIFAR-10 dataset and monophonic music sequences. Notably, we find our tau-leaping with predictor-corrector sampler can provide higher quality CIFAR10 samples than previous discrete time discrete state approaches, further closing the performance gap between when images are modeled as discrete data or as continuous data.

Proofs for all propositions and theorems are given in the Appendix.

2 Background on Discrete Denoising Models

In the discrete time, discrete state space case, we aim to model discrete data $x_0 \in \mathcal{X}$ with finite cardinality $S = |\mathcal{X}|$. We assume $x_0 \sim p_{\text{data}}(x_0)$ for some discrete data distribution $p_{\text{data}}(x_0)$. We define a forward noising process that transforms $p_{\text{data}}(x_0)$ to some distribution $q_K(x_K)$ that closely approximates an easy to sample distribution $p_{\text{ref}}(x_K)$. This is done by defining forward kernels $q_{k+1|k}(x_{k+1}|x_k)$ that all admit p_{ref} as a stationary distribution and mix reasonably quickly. For example, one can use a simple uniform kernel [6, 8], $q_{k+1|k}(x_{k+1}|x_k) = \delta_{x_{k+1},x_k}(1-\beta) + (1-\delta_{x_{k+1},x_k})\beta/(S-1)$ where δ is a Kronecker delta. The corresponding p_{ref} is the uniform distribution over all states. Other choices include: an absorbing state kernel—where for each state there is a small probability that it transitions to some absorbing state—or a discretized Gaussian kernel—where only transitions to nearby states have significant probability (valid for spaces with ordinal structure) [8].

After defining $q_{k+1|k}$, we have a forward joint decomposition as follows

$$q_{0:K}(x_{0:K}) = p_{\text{data}}(x_0) \prod_{k=0}^{K-1} q_{k+1|k}(x_{k+1}|x_k).$$

The joint distribution $q_{0:K}(x_{0:K})$ also admits a reverse decomposition:

$$q_{0:K}(x_{0:K}) = q_K(x_K) \prod_{k=0}^{K-1} q_{k|k+1}(x_k|x_{k+1}) \text{ where } q_{k|k+1}(x_k|x_{k+1}) = \frac{q_{k+1|k}(x_{k+1}|x_k) q_k(x_k)}{q_{k+1}(x_{k+1})}.$$

Here $q_k(x_k)$ denotes the marginal of $q_{0:K}(x_{0:K})$ at time k. If one had access to $q_{k|k+1}$ and could sample q_K exactly, then samples from $p_{\text{data}}(x_0)$ could be produced by first sampling $x_K \sim q_K(\cdot)$ and then ancestrally sampling the reverse kernels, i.e. $x_k \sim q_{k|k+1}(\cdot|x_{k+1})$.

However, in practice, $q_{k|k+1}$ is intractable and needs to be approximated with a parametric reverse kernel, $p^\theta_{k|k+1}$. This kernel is commonly defined through the analytic $q_{k|k+1,0}$ distribution and a parametric 'denoising' model $p^\theta_{0|k+1}$ [6, 8],

$$\begin{aligned} p^\theta_{k|k+1}(x_k|x_{k+1}) &\triangleq \textstyle\sum_{x_0} q_{k|k+1,0}(x_k|x_{k+1},x_0)p^\theta_{0|k+1}(x_0|x_{k+1}) \\ &= q_{k+1|k}(x_{k+1}|x_k)\textstyle\sum_{x_0}\frac{q_{k|0}(x_k|x_0)}{q_{k+1|0}(x_{k+1}|x_0)}p^\theta_{0|k+1}(x_0|x_{k+1}). \end{aligned} \tag{1}$$

Though $q_K(x_K)$ is also intractable, for large K we can reliably approximate it with $p_{\text{ref}}(x_K)$. Note that the faster the transitions mix, the more accurate this approximation becomes. Approximate samples from $p_{\text{data}}(x_0)$ can then be obtained by sampling the generative joint distribution

$$p^\theta_{0:K}(x_{0:K}) = p_{\text{ref}}(x_K)\textstyle\prod_{k=0}^{K-1} p^\theta_{k|k+1}(x_k|x_{k+1}),$$

where θ is trained through minimizing the negative discrete time (DT) ELBO which is an upper bound on the negative model log-likelihood

$$\mathbb{E}_{p_{\text{data}}(x_0)}\left[-\log p^\theta_0(x_0)\right] \leq \mathbb{E}_{q_{0:K}(x_{0:K})}\left[-\log\frac{p^\theta_{0:K}(x_{0:K})}{q_{1:K|0}(x_{1:K}|x_0)}\right] = \mathcal{L}_{\text{DT}}(\theta).$$

It was shown in [1] that $\mathcal{L}_{\text{DT}}$ can be re-written as

$$\begin{aligned} \mathcal{L}_{\text{DT}}(\theta) = \mathbb{E}_{p_{\text{data}}(x_0)}\Big[&\text{KL}(q_{K|0}(x_K|x_0)||p_{\text{ref}}(x_K)) - \mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p^\theta_{0|1}(x_0|x_1)\right] \\ &+ \textstyle\sum_{k=1}^{K-1}\mathbb{E}_{q_{k+1|0}(x_{k+1}|x_0)}\left[\text{KL}(q_{k|k+1,0}(x_k|x_{k+1},x_0)||p^\theta_{k|k+1}(x_k|x_{k+1}))\right]\Big] \end{aligned}$$

where KL is the Kullback–Leibler divergence. The forward kernels $q_{k+1|k}$ are chosen such that $q_{k|0}(x_k|x_0)$ can be computed efficiently in a time independent of k. With this, θ can be efficiently trained by taking a random selection of terms from $\mathcal{L}_{\text{DT}}$ in each minibatch and performing a stochastic gradient step.

3 Continuous Time Framework

3.1 Forward process and its time reversal

Our method is built upon a continuous time process from $t = 0$ to $t = T$. State transitions can occur at any time during this process as opposed to the discrete time case where transitions only occur when one of the finite number of transition kernels is applied (see Figure 1). This process is known as a Continuous Time Markov Chain (CTMC), we provide a short overview of CTMCs in Appendix A for completeness. Giving an intuitive introduction here, we can define a CTMC through an initial distribution q_0 and a transition rate matrix $R_t \in \mathbb{R}^{S\times S}$. If the current state is $\tilde{x}$, then the transition rate matrix entry $R_t(\tilde{x}, x)$ is the instantaneous rate (occurrences per unit time) at which state $\tilde{x}$ transitions to state x. Loosely speaking, the next state in the process will likely be one for which $R_t(\tilde{x}, x)$ is high, and furthermore, the higher the rate is, the less time it will take for this transition to occur.

It turns out that the transition rate, R_t, also defines the infinitesimal transition probability for the process between the two time points $t - \Delta t$ and t

$$q_{t|t-\Delta t}(x|\tilde{x}) = \delta_{x,\tilde{x}} + R_t(\tilde{x}, x)\Delta t + o(\Delta t),$$

where $o(\Delta t)$ represents terms that tend to zero at a faster rate than Δt. Comparing to the discrete time case, we see that R_t assumes an analogous role to the discrete time forward kernel $q_{k+1|k}$ in how we define the forward process. Therefore, just as in discrete time, we design R_t such that: i) the forward process mixes quickly towards an easy to sample (stationary) distribution, p_{ref}, (e.g. uniform), ii) we can analytically obtain $q_{t|0}(x_t|x_0)$ distributions to enable efficient training (see Section 4.1 for how this is done). We initialize the forward CTMC at $q_0(x_0) = p_{\text{data}}(x_0)$ at time $t = 0$. We denote the marginal at time $t = T$ as $q_T(x_T)$, which should be close to $p_{\text{ref}}(x_T)$.

We now consider the time reversal of the forward process, which will take us from the marginal $q_T(x_T)$ back to the data distribution $p_{\text{data}}(x_0)$ through a reverse transition rate matrix, $\hat{R}_t \in \mathbb{R}^{S\times S}$:

$$q_{t|t+\Delta t}(\tilde{x}|x) = \delta_{\tilde{x},x} + \hat{R}_t(x, \tilde{x})\Delta t + o(\Delta t).$$

In discrete time, one uses Bayes rule to go from $q_{k+1|k}$ to $q_{k|k+1}$. We can use similar ideas to calculate $\hat{R}_t$ from R_t as per the following result.

Proposition 1. *For a forward in time CTMC, $\{x_t\}_{t\in[0,T]}$, with rate matrix R_t, initial distribution $p_{\text{data}}(x_0)$ and terminal distribution $q_T(x_T)$, there exists a CTMC with initial distribution $q_T(x_T)$ at $t = T$, terminal distribution $p_{\text{data}}(x_0)$ at $t = 0$ and transition rate matrix $\hat{R}_t$ that runs backwards in time and is almost everywhere equivalent to the time reversal of the forward CTMC, $\{x_t\}_{t\in[T,0]}$. Furthermore, $\hat{R}_t$ is related to R_t by the following expression*

$$\hat{R}_t(x, \tilde{x}) = R_t(\tilde{x}, x) \sum_{x_0} \frac{q_{t|0}(\tilde{x}|x_0)}{q_{t|0}(x|x_0)} q_{0|t}(x_0|x) \quad \textit{for} \quad x \neq \tilde{x},$$

where $q_{t|0}(x|x_0)$ are the conditional marginals of the forward process and $q_{0|t}(x_0|x) = q_{t|0}(x|x_0)p_{\text{data}}(x_0)/q_t(x)$ with $q_t(x)$ being the marginal of the forward process at time t. When $x = \tilde{x}$, $\hat{R}_t(x, x) = -\sum_{x'\neq x} \hat{R}_t(x, x')$ because the rows must sum to zero (see Appendix A).

Unfortunately, $\hat{R}_t$ is intractable due to the intractability of $q_t(x)$ and thus of $q_{0|t}(x_0|x)$. Therefore, we consider an approximation $\hat{R}^\theta_t$ of $\hat{R}_t$ by approximating $q_{0|t}(x_0|x)$ with a parametric denoising model, $p^\theta_{0|t}(x_0|x)$:

$$\hat{R}^\theta_t(x, \tilde{x}) = R_t(\tilde{x}, x) \sum_{x_0} \frac{q_{t|0}(\tilde{x}|x_0)}{q_{t|0}(x|x_0)} p^\theta_{0|t}(x_0|x) \quad \text{for} \quad x \neq \tilde{x}$$

and $\hat{R}^\theta_t(x, x) = -\sum_{x'\neq x} \hat{R}^\theta_t(x, x')$ as before. As a further analogy to the discrete time case, notice that when $x \neq \tilde{x}$, $\hat{R}^\theta_t$ has the same form as the discrete time parametric reverse kernel, $p^\theta_{k|k+1}$ defined in eq (1) but with the forward kernel, $q_{k+1|k}$, replaced by the forward rate, R_t.

3.2 Continuous Time ELBO

In discrete time, θ is trained by minimizing the discrete time negative ELBO, $\mathcal{L}_{\text{DT}}$, formed from the forward and reverse processes. We mirror this approach in continuous time by minimizing the corresponding continuous time (CT) negative ELBO, $\mathcal{L}_{\text{CT}}$, as derived below.

Proposition 2. *For the reverse in time CTMC with initial distribution $p_{ref}(x_T)$, terminal distribution $p^\theta_0(x_0)$, and reverse rate $\hat{R}^\theta_t$, an upper bound on the negative model log-likelihood, $\mathbb{E}_{p_{\text{data}}(x_0)}[-\log p^\theta_0(x_0)]$, is given by*

$$\mathcal{L}_{\text{CT}}(\theta) = T\, \mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(x)r_t(\tilde{x}|x)} \left[\left\{ \sum_{x'\neq x} \hat{R}^\theta_t(x, x') \right\} - \mathcal{Z}^t(x) \log \left(\hat{R}^\theta_t(\tilde{x}, x) \right) \right] + C,$$

where C is a constant independent of θ and

$$\mathcal{Z}^t(x) = \sum_{x'\neq x} R_t(x, x') \qquad\qquad r_t(\tilde{x}|x) = (1 - \delta_{\tilde{x},x}) R_t(x, \tilde{x})/\mathcal{Z}^t(x).$$

Here $r_t(\tilde{x}|x)$ gives the probability of transitioning from x to $\tilde{x}$, given that we know a transition occurs at time t. We can optimize this objective efficiently with stochastic gradient descent. For a gradient update, we sample a batch of datapoints from $p_{\text{data}}(x_0)$, noise each datapoint using a random time, $t \sim \mathcal{U}(0, T)$, $x \sim q_{t|0}(x|x_0)$ and finally sample an auxiliary $\tilde{x}$ from $r_t(\tilde{x}|x)$ for each x. Intuitively, $(x, \tilde{x})$ are a pair of states following the forward in time noising process. Minimizing the second term in $\mathcal{L}_{\text{CT}}$ maximizes the reverse rate for this pair, but going in the backwards direction, $\tilde{x}$ to x. This is how $\hat{R}^\theta_t$ learns to reverse the noising process. Intuition on the first term and a direct comparison to $\mathcal{L}_{\text{DT}}$ is given in Appendix C.1.

The first argument of $\hat{R}^\theta_t$ is input into $p^\theta_{0|t}$ so we naively require two network forward passes on x and $\tilde{x}$ to evaluate the objective. We can avoid this by approximating the $q_t(x)$ sample in the first term with $\tilde{x}$ meaning we need only evaluate the network once on $\tilde{x}$. The approximation is valid because, as we show in Appendix C.4, $\tilde{x}$ is approximately distributed according to $q_{t+\delta t}$ for δt very small.

4 Efficient Forward and Backward Sampling

4.1 Choice of Forward Process

The transition rate matrix R_t needs to be chosen such that the forward process: i) mixes quickly towards p_{ref}, and ii) the $q_{t|0}(x|x_0)$ distributions can be analytically obtained. The Kolmogorov

differential equation for the CTMC needs to be integrated to obtain $q_{t|0}(x|x_0)$. This can be done analytically when R_t and $R_{t'}$ commute for all t, t', see Appendix E. An easy way to meet this condition is to let $R_t = \beta(t)R_b$ where $R_b \in \mathbb{R}^{S\times S}$ is a user-specified time independent base rate matrix and $\beta(t) \in \mathbb{R}$ is a time dependent scalar. We then obtain the analytic expression

$$q_{t|0}(x = j|x_0 = i) = \left(Q\exp\left[\Lambda \int_0^t \beta(s)ds\right] Q^{-1}\right)_{ij}$$

where $R_b = Q\Lambda Q^{-1}$ is the eigendecomposition of matrix R_b and $\exp[\cdot]$ the element-wise exponential.

Our choice of β schedule is guided by [3, 4], $\beta(t) = ab^t \log(b)$. The hyperparameters a and b are selected such that $q_T(x) \approx p_{\text{ref}}(x)$ at the terminal time $t = T$ while having a steady speed of 'information corruption' which ensures that $\hat{R}_t$ does not vary quickly in a short span of time.

We experiment with a variety of R_b matrices, for example, a uniform rate, $R_b = \mathbb{1}\mathbb{1}^T - S\text{Id}$, where $\mathbb{1}\mathbb{1}^T$ is a matrix of ones and Id is the identity. For problems with a heavy spatial bias, e.g. images, we can instead use a forward rate that only encourages transitions to nearby states; details and the links to the corresponding discrete time processes can be found in Appendix E.

4.2 Factorizing Over Dimensions

Our aim is to model data that is D dimensional, with each dimension taking one value from S possibilities. We now slightly redefine notation and say $\boldsymbol{x}^{1:D} \in \mathcal{X}^D$, $|\mathcal{X}| = S$. In this setting, calculating transition probabilities naively would require calculating S^D rate values corresponding to each of the possible next states. This is intractable for any reasonably sized S and D. We avoid this problem simply by factorizing the forward process such that each dimension propagates independently. Since this is a continuous time process and each dimension's forward process is independent of the others, the probability two or more dimensions transition at exactly the same time is zero. Therefore, overall in the full dimensional forward CTMC, each transition only ever involves a change in exactly one dimension. For the time reversal CTMC, it will also be true that exactly one dimension changes in each transition. This makes computation tractable because of the S^D rate values, only $D \times (S-1) + 1$ are non-zero - those corresponding to transitions where exactly one dimension changes plus the no change transition. Finally, we note that even though dimensions propagate independently in the forward direction, they are not independent in the reverse direction because the starting points for each dimension's forward process are not independent for non factorized p_{data}. The following proposition shows the exact forms for the forward and reverse rates in this case.

Proposition 3. *If the forward process factorizes as $q_{t|s}(\boldsymbol{x}_t^{1:D}|\boldsymbol{x}_s^{1:D}) = \prod_{d=1}^D q_{t|s}(x_t^d|x_s^d)$, $t > s$, then the forward and reverse rates are of the form*

$$R_t^{1:D}(\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}) = \textstyle\sum_{d=1}^D R_t^d(\tilde{x}^d, x^d)\delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}},$$

$$\hat{R}_t^{1:D}(\boldsymbol{x}^{1:D}, \tilde{\boldsymbol{x}}^{1:D}) = \textstyle\sum_{d=1}^D R_t^d(\tilde{x}^d, x^d)\delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}^{1:D}) \frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)},$$

where $R_t^d \in \mathbb{R}^{S\times S}$ and $\delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}}$ is 1 when all dimensions except for d are equal.

To find $\hat{R}_t^{\theta\,1:D}$ we simply replace $q_{0|t}(x_0^d|\boldsymbol{x}^{1:D})$ with $p_{0|t}^\theta(x_0^d|\boldsymbol{x}^{1:D})$ which is easily modeled with a neural network that outputs conditionally independent state probabilities in each dimension. In Appendix C.3 we derive the form of $\mathcal{L}_{\text{CT}}$ when we use this factorized form for $R_t^{1:D}$ and $\hat{R}_t^{\theta\,1:D}$.

4.3 Simulating the Generative Reverse Process with Tau-Leaping

The parametric generative reverse process is a CTMC with rate matrix $\hat{R}_t^{\theta\,1:D}$. Simulating this process from distribution $p_{\text{ref}}(\boldsymbol{x}_T^{1:D})$ at time $t = T$ back to $t = 0$ will produce approximate samples from $p_{\text{data}}(\boldsymbol{x}_0^{1:D})$. The process could be simulated exactly using Gillespie's Algorithm [21, 22, 23] which alternates between i) sampling a holding time to remain in the current state and ii) sampling a new state according to the current rate matrix, $\hat{R}_t^{\theta\,1:D}$ (see Appendix F). This is inefficient for large D because we would need to step through each transition individually and so only one dimension would change for each simulation step.

Instead, we use tau-leaping [20, 23], a very popular approximate simulation method developed in chemical physics. Rather than step back through time one transition to the next, tau-leaping leaps

from t to $t-\tau$ and applies all transitions that occurred in $[t-\tau, t]$ simultaneously. To make a leap, we assume $\hat{R}_t^{\theta\,1:D}$ and $\boldsymbol{x}_t^{1:D}$ remain constant in $[t-\tau, t]$. As we propagate from t to $t-\tau$, we count all of the transitions that occur, but hold off on actually applying them until we reach $t-\tau$, such that $\boldsymbol{x}_t^{1:D}$ remains constant in $[t-\tau, t]$. Assuming $\hat{R}_t^{\theta\,1:D}$ and $\boldsymbol{x}_t^{1:D}$ remain constant, the number of times a transition from $\boldsymbol{x}_t^{1:D}$ to $\tilde{\boldsymbol{x}}^{1:D}$ occurs in $[t-\tau, t]$ is Poisson distributed with mean $\tau \hat{R}_t^{\theta\,1:D}(\boldsymbol{x}_t^{1:D}, \tilde{\boldsymbol{x}}^{1:D})$. Once we reach $t-\tau$, we apply all transitions that occurred simultaneously i.e. $\boldsymbol{x}_{t-\tau}^{1:D} = \boldsymbol{x}_t^{1:D} + \sum_i P_i(\tilde{\boldsymbol{x}}_i^{1:D} - \boldsymbol{x}_t^{1:D})$ where P_i is a Poisson random variable with mean $\tau \hat{R}_t^{\theta\,1:D}(\boldsymbol{x}_t^{1:D}, \tilde{\boldsymbol{x}}_i^{1:D})$. Note the sum assumes a mapping from $\mathcal{X}$ to $\mathbb{Z}$.

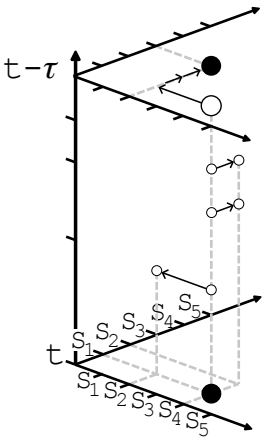


Figure 2: 3D visualization of one tau-leaping step from $x_t^{1:2} = \{S_4, S_1\}$ to $x_{t-\tau}^{1:2} = \{S_2, S_3\}$. Here, $D = 2$, $|\mathcal{X}| = 5$, $P_{12} = 1$, $P_{22} = 2$, all other $P_{ds} = 0$.

Using our knowledge of $\hat{R}_t^{\theta\,1:D}$, we can further unpack this update. Namely, $\hat{R}_t^{\theta\,1:D}(\boldsymbol{x}_t^{1:D}, \tilde{\boldsymbol{x}}^{1:D})$ can only be non-zero when $\tilde{\boldsymbol{x}}^{1:D}$ has a different value to $\boldsymbol{x}_t^{1:D}$ in exactly one dimension (rates for multi-dimensional changes are zero). Explicitly summing over these options we get $\boldsymbol{x}_{t-\tau}^{1:D} = \boldsymbol{x}_t^{1:D} + \sum_{d=1}^{D} \sum_{s=1\setminus x_t^d}^{S} P_{ds}(s - x_t^d)\boldsymbol{e}^d$ where $\boldsymbol{e}^d$ is a one-hot vector with a 1 at dimension d and P_{ds} is a Poisson random variable with mean $\tau \hat{R}_t^{\theta\,1:D}(\boldsymbol{x}_t^{1:D}, \boldsymbol{x}_t^{1:D} + (s - x_t^d)\boldsymbol{e}^d)$. Since multiple P_{ds} can be non-zero, we see that tau-leaping allows $\boldsymbol{x}_t^{1:D}$ to change in multiple dimensions in a single step. Figure 2 visualizes this idea. During the $[t-\tau, t]$ interval, one jump occurs in dimension 1 and two jumps occur in dimension 2. These are all applied simultaneously once we reach $t-\tau$. When our discrete data has ordinal structure (e.g. Section 6.2) our mapping to $\mathbb{Z}$ is not arbitrary and making multiple jumps within the same dimension ($\sum_{s=1\setminus x_t^d}^{S} P_{ds} > 1$) is meaningful. In the non-ordinal/categorical case (e.g. Section 6.3) the mapping to $\mathbb{Z}$ is arbitrary and so, although taking simultaneous jumps in different dimensions is meaningful, taking multiple jumps within the same dimension is not. For this type of data, we reject changes to x_t^d for any d for which $\sum_{s=1\setminus x_t^d}^{S} P_{ds} > 1$. In practice, the rejection rate is very small when $\hat{R}_t^{1:D}$ is suitable for categorical data (e.g. uniform), see Appendix H.3. In Section 4.5, our error bound accounts for this low probability of rejection and also the low probability of an out of bounds jump that we observe in practice in the ordinal case.

The tau-leaping approximation improves with smaller τ, recovering exact simulation in the limit as $\tau \to 0$. Exact simulation is similar to an autoregressive model in that only one dimension changes per step. Increasing τ and thus the average number of dimensions changing per step gives us a natural way to modulate the 'autoregressiveness' of the model and trade sample quality with compute (Figure 4 right). We refer to our method of using tau-leaping to simulate the reverse CTMC as τLDR (tau-leaping denoising reversal) which we formalize in Algorithm 1 in Appendix F.

We note that theoretically, one could approximate $\hat{R}_t^{\theta\,1:D}$ as constant in the interval $[t-\tau, t]$, and construct a transition probability matrix by solving the forward Kolmogorov equation with the matrix exponential $P_{t-\tau|t} \approx \exp(\tau \hat{R}_t^{\theta\,1:D})$. However, for the learned $\hat{R}_t^{\theta\,1:D} \in \mathbb{R}^{S^D \times S^D}$ matrix, it is intractable to compute this matrix exponential so we use tau-leaping for sampling instead.

4.4 Predictor-Corrector

During approximate reverse sampling, we aim for the marginal distribution of samples at time t to be close to $q_t(x_t)$ (the marginal at time t of the true CTMC). The continuous time framework allows us to exploit additional information to more accurately follow the reverse progression of marginals, $\{q_t(x_t)\}_{t\in[T,0]}$ and improve sample quality. Namely, after a tau-leaping 'predictor' step using rate $\hat{R}_t^{\theta}$, we can apply 'corrector' steps with rate R_t^c which has $q_t(x_t)$ as its stationary distribution. The corrector steps bring the distribution of samples at time t closer to the desired $q_t(x_t)$ marginal. R_t^c is easy to calculate as stated below

Proposition 4. *For a forward CTMC with marginals $\{q_t(x_t)\}_{t\in[0,T]}$, forward rate, R_t, and corresponding reverse CTMC with rate $\hat{R}_t$, the rate $R_t^c = R_t + \hat{R}_t$ has $q_t(x_t)$ as its stationary distribution.*

In practice, we approximate R_t^c by replacing $\hat{R}_t$ with $\hat{R}_t^\theta$. This is directly analogous to Predictor-Corrector samplers in continuous state spaces [4] that predict by integrating the reverse SDE and correct with score-based Markov chain Monte Carlo steps, see Appendix F.2 for further discussion.

4.5 Error Bound

Our continuous time framework also allows us to provide a novel theoretical bound on the error between the true data distribution and the sample distribution generated via tau-leaping (without predictor-corrector steps), in terms of the error in our approximation of the reverse rate and the mixing of the forward noising process.

We assume we have a time-homogeneous rate matrix R_t on $\mathcal{X}$, from which we construct the factorized rate matrix $R_t^{1:D}$ on $\mathcal{X}^D$ by setting $R_t^d = R_t$ for each d. Note that by rescaling time by a factor of $\beta(t)$ we can transform our choice of rate from Section 4.1 to be time-homogeneous. We will denote $|R| = \sup_{t\in[0,T],x\in\mathcal{X}} |R_t(x,x)|$, and let t_{mix} be the (1/4)-mixing time of the CTMC with rate R_t (see [24, Chapter 4.5]).

Theorem 1. *For any $D \geq 1$ and distribution p_{data} on $\mathcal{X}^D$, let $\{x_t\}_{t\in[0,T]}$ be a CTMC starting in p_{data} with rate matrix $R_t^{1:D}$ as above. Suppose that $\hat{R}_t^{\theta\,1:D}$ is an approximation to the reverse rate matrix and let $(y_k)_{k=0,1,\dots,N}$ be a tau-leaping approximation to the reverse dynamics with maximum step size τ. Suppose further that there is some constant $M > 0$ independent of D such that*

$$\sum_{y\neq x} \left| \hat{R}_t^{1:D}(x,y) - \hat{R}_t^{\theta\,1:D}(x,y) \right| \leq M \tag{2}$$

for all $t \in [0,T]$. Then under the assumptions in Appendix B.5, there are constants $C_1, C_2 > 0$ depending on $\mathcal{X}$ and R_t but not D such that, if $\mathcal{L}(y_0)$ denotes the law of y_0, we have the total variation bound

$$||\mathcal{L}(y_0) - p_{\text{data}}||_{\text{TV}} \leq 3MT + \left\{ \left(|R|SDC_1\right)^2 + \tfrac{1}{2}C_2(M + C_1SD|R|) \right\} \tau T + 2\exp\left\{ -\frac{T\log^2 2}{t_{\text{mix}}\log 4D} \right\}$$

The first term of the above bound captures the error introduced by our approximation of the reverse rate $\hat{R}_t^{1:D}$ with $\hat{R}_t^{\theta\,1:D}$. The second term reflects the error introduced by the tau-leaping approximation, and is linear in both T and τ, showing that as we take our tau-leaping steps to be arbitrarily small, the error introduced by tau-leaping goes to zero. The final term describes the mixing of the forward chain, and captures the error introduced since p_{ref} and q_T are not exactly equal.

We choose to make the dependence of the bound on the dimension D explicit, since we are specifically interested in applying tau-leaping to high dimensional problems where we make transitions in different dimensions simultaneously in a single time step. The bound grows at worst quadratically in the dimension, versus e.g. exponentially. The bound is therefore useful in showing us that we do not need to make τ impractically small in high dimensions. Other than gaining these intuitions, we do not expect the bound to be particularly tight in practice and further it would not be practical to compute because of the difficulty in finding M, C_1 and C_2.

The assumptions listed in Appendix B.5 hold approximately for tau-leaping in practice when we use spatially biased rates for ordinal data such that jump sizes are small or uniform rates for non-ordinal data such that the dimensional rejection rate is small. These assumptions could be weakened, however, Theorem 1 would become much more involved, obscuring the intuition and structure of the problem.

5 Related Work

The application of denoising models to discrete data was first described in [1] using a binomial diffusion process for a binary dataset. Each reverse kernel $p_{k|k+1}^\theta$ was directly parameterized without using a denoising model $p_{0|k}^\theta$. In [25] an approach for discrete categorical data was suggested using a uniform forward noising kernel, $q_{k+1|k}$, and a reverse kernel parameterized through a denoising model, though no experiments were performed with the approach. Experiments on text and segmentation maps were then performed with a similar model in [6]. Other forward kernels were introduced in [8] that are more appropriate for certain data types such as the spatially biased Gaussian kernel. [9, 13] apply the approach to discrete latent space modeling using uniform and absorbing state forward

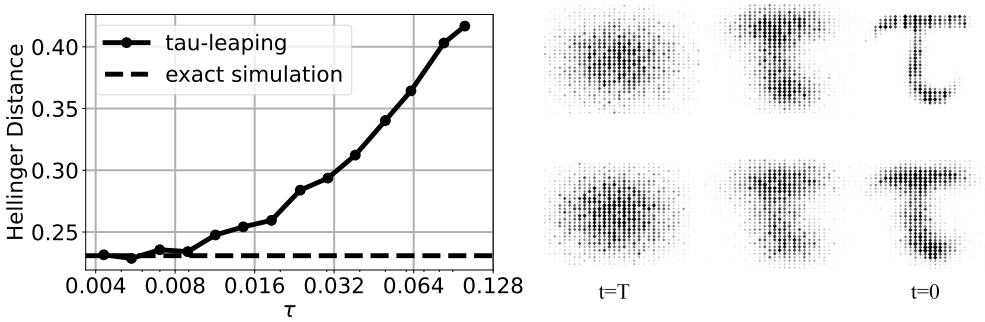


Figure 3: **Left:** Hellinger distance between the true training distribution and generated sample distributions with exact simulation or tau-leaping. With τ small, we simulate the reverse CTMC with the same fidelity as the exact simulation. **Top Right:** Histograms of the marginals during the reverse generative process simulated using tau-leaping with $\tau = 0.004$. Darker and larger diamonds represent increased density. **Bottom Right:** The same for $\tau = 0.1$, note the reduced sample quality.

kernels. Whilst a link to continuous time for the forward process is mentioned in [8], all of these approaches train and sample in discrete time. We show in Appendix G that this involves making an implicit approximation for multi-dimensional data. We extend this line of work by training and sampling in continuous time.

Other works also operate in discrete space but less rigidly follow the diffusion framework. A corruption process tailored to text is proposed in [12], whereby token deletion and insertion is also incorporated. [26] also focus on text, creating a generative reverse chain that repeatedly applies the same denoising kernel. The corruption distribution is also defined through the same denoising kernel to reduce distribution shift between training and sampling. In [7], a more standard masking based forward process is used but the reversal is interpreted from an order agnostic autoregressive perspective. They also describe how their model can be interpreted as the reversal of a continuous time absorbing state diffusion but do not utilize this perspective in training or sampling. [27] propose a denoising type framework that can be used on binary data where the forward and reverse process share the same transition kernel. Finally, in [11], the discrete latent space of a VQVAE is modeled by quantizing an underlying continuous state space diffusion with probabilistic quantization functions.

6 Experiments

6.1 Demonstrative Example

We first verify the method can accurately produce samples from the entire support of the data distribution and that tau-leaping can accurately simulate the reverse CTMC. To do this, we create a dataset formed of 2d samples of a state space of 32 arranged such that the histogram of the training dataset forms a 'τ' shape. We train a denoising model using the $\mathcal{L}_{\text{CT}}$ objective with $p^{\theta}_{0|t}$ parameterized through a residual MLP (full details in Appendix H.1). We then sample the parameterized reverse process using an exact method (up to needing to numerically integrate the reverse rate) and tau-leaping. Figure 3 top-right shows the marginals during reverse simulation with $\tau = 0.004$ and we indeed produce samples from the entire support of p_{data}. Furthermore, we find that with sufficiently small τ, we can match the fidelity of exact simulation of the reverse CTMC (Figure 3 left). The value of τ dictates the number of network evaluations in the reverse process according to NFE $= T/\tau$. In all experiments we use $T = 1$. Exact simulation results in a non zero Hellinger distance between the generated and training distributions because of imperfections in the learned $\hat{R}^{\theta}_t$ model.

6.2 Image Modeling

We now demonstrate that our continuous time framework gives us improved generative modeling performance versus operating in discrete time. We show this on the CIFAR-10 image dataset. Images are typically stored as discrete data, each pixel channel taking one value from 256 possibilities. Continuous state space methods have to somehow get around this fact by, for example, adding a discretization function at the end of the generative process [3] or adding uniform noise to the data.

Table 1: Sample quality metrics and model likelihoods for diffusion methods modeling CIFAR10 in discrete state space. Diffusion methods modeling CIFAR10 in continuous space are included for reference. The Inception Score (IS) and Fréchet Inception Distance (FID) are calculated using 50000 generated samples with respect to the training dataset as is standard practice. The ELBO values are reported on the test set in bits per dimension.

	Method	IS (↑)	FID (↓)	ELBO (↑)
Discrete state	D3PM Absorbing [8]	6.78	30.97	-4.40
	D3PM Gauss [8]	8.56	7.34	$\mathbf{-3.44}$
	τLDR-0 (ours)	8.74	8.10	-3.59
	τLDR-10 (ours)	**9.49**	**3.74**	-3.59
Continuous state	DDPM [3]	9.46	3.17	-3.75
	NCSN [4]	9.89	2.20	-

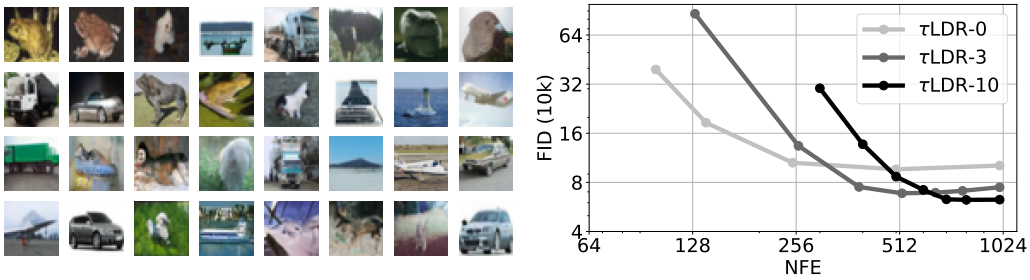


Figure 4: **Left:** Unconditional CIFAR10 samples from our τLDR-10 model **Right:** FID scores for the generated CIFAR10 samples versus number of $p^\theta_{0|t}$ evaluations during sampling (variation induced by varying τ). Calculated with 10k samples, hence the discrepancy with Table 1 [28].

Here, we model the images directly in discrete space. We parameterize $p^\theta_{0|t}$ using the standard U-net architecture [3] with the modifications for discrete state space suggested by [8]. We use a spatially biased rate matrix and train with an augmented $\mathcal{L}_{\text{CT}}$ loss including direct $p^\theta_{0|t}$ supervision, full experimental details are in Appendix H.2.

Figure 4 left shows randomly generated unconditional CIFAR10 samples from the model and we report sample quality metrics in Table 1. We see that our method (τLDR-0) with 0 corrector steps has better Inception Score but worse FID than the D3PM discrete time method. However, our τLDR-10 method with 10 corrector steps per predictor step at the end of the reverse sampling process ($t < 0.1T$) greatly improves sample quality, beating the discrete time method in both metrics and further closes the performance gap with methods modeling images as continuous data. The derivation of the corrector rate which gave us this improved performance required our continuous time framework. D3PM achieves the highest ELBO but we note that this does not correlate well with sample quality. In Table 1, τ was adjusted such that both τLDR-0 and τLDR-10 used 1000 $p^\theta_{0|t}$ evaluations in the reverse sampling procedure. We show how FID score varies with number of $p^\theta_{0|t}$ evaluations for τLDR-$\{0, 3, 10\}$ in Figure 4 right. The optimum number of corrector steps depends on the sampling budget, with lower numbers of corrector steps being optimal for tighter budgets. This is due to the increased τ required to maintain a fixed budget when we use a larger number of corrector steps.

6.3 Monophonic Music

In this experiment, we demonstrate our continuous time model improves generation quality on non-ordinal/categorical discrete data. We model songs from the Lakh pianoroll dataset [29, 30]. We select all monophonic sequences from the dataset such that at each of the 256 time steps either one from 128 notes is played or it is a rest. Therefore, our data has state space size $S = 129$ and dimension $D = 256$. We scramble the ordering of the state space when mapping to $\mathbb{Z}$ to destroy any ordinal structure. We parameterize $p^\theta_{0|t}$ with a transformer architecture [31] and train using a conditional form of $\mathcal{L}_{\text{CT}}$ targeting the conditional distribution of the final 14 bars (224 time steps) given the first 2 bars of the song. We use a uniform forward rate matrix, R_t, full experimental details

Table 2: Metrics comparing generated conditional samples and ground truth completions. We compute these over the test set showing mean±std with respect to 5 samples for each test song.

Model	Hellinger Distance	Proportion of Outliers
τLDR-0 Birth/Death	0.3928 ± 0.0010	0.1316 ± 0.0012
τLDR-0 Uniform	0.3765 ± 0.0013	0.1106 ± 0.0010
τLDR-2 Uniform	$\mathbf{0.3762 \pm 0.0015}$	$\mathbf{0.1091 \pm 0.0014}$
D3PM Uniform [8]	0.3839 ± 0.0002	0.1137 ± 0.0010

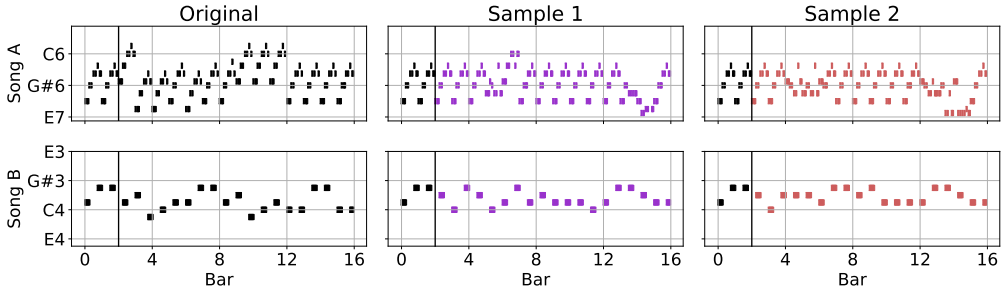


Figure 5: Conditional completions of an unseen music sequence. The conditioning 2 bars are shown to the left of the black line. More examples and audio recordings are linked in Appendix H.3.

are given in Appendix H.3. Conditional completions of unseen test songs are shown in Figure 5. The model is able to faithfully complete the piece in the same style as the conditioning bars.

We quantify sample quality in Table 2. We use two metrics: the Hellinger distance between the histograms of generated and ground truth notes and the proportion of outlier notes in the generations but not in the ground truth. Using our method, we compare between a birth/death and uniform forward rate matrix R_t. The birth/death rate is only non-zero for adjacent states whereas the uniform rate allows transitions between arbitrary states which is more appropriate for the categorical case thus giving improved sample quality. Adding 2 corrector steps per predictor step further improves sample quality. We also compare to the discrete time method D3PM [8] with its most suitable corruption process for categorical data. We find it performs worse than our continuous time method.

7 Discussion

We have presented a continuous time framework for discrete denoising models. We showed how to efficiently sample the generative process with tau-leaping and provided a bound on the error of the generated samples. On discrete data problems, we found our predictor-corrector sampler improved sample quality versus discrete time methods. Regarding limitations, our model requires many model evaluations to produce a sample. Our work has opened the door to applying the work improving sampling speed on continuous data [14, 15, 16, 17, 32] to discrete data problems too. Modeling performance on images is also slightly behind continuous state space models, we hope this gap is further closed with bespoke discrete state architectures and corruption process tuning. Finally, we note that the ELBO values for the discrete time model on CIFAR10 are better than for our method. In this work, we focused on sample quality rather than using our model to give data likelihoods e.g. for compression downstream tasks.

Acknowledgements

Andrew Campbell and Joe Benton acknowledge support from the EPSRC CDT in Modern Statistics and Statistical Machine Learning (EP/S023151/1). Arnaud Doucet is partly supported by the EPSRC grant EP/R034710/1. He also acknowledges support of the UK Defence Science and Technology Laboratory (DSTL) and EPSRC under grant EP/R013616/1. This is part of the collaboration between US DOD, UK MOD and UK EPSRC under the Multidisciplinary University Research Initiative. This project made use of time on Tier 2 HPC facility JADE2, funded by EPSRC (EP/T022205/1).

References

[1] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning*, 2015.

[2] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 2019.

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020.

[4] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.

[5] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 2021.

[6] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 2021.

[7] Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *International Conference on Learning Representations*, 2022.

[8] Jacob Austin, Daniel Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 2021.

[9] Patrick Esser, Robin Rombach, Andreas Blattmann, and Bjorn Ommer. Imagebart: Bidirectional context with multinomial diffusion for autoregressive image synthesis. *Advances in Neural Information Processing Systems*, 2021.

[10] Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. *International Conference on Machine Learning*, 2022.

[11] Max Cohen, Guillaume Quispe, Sylvain Le Corff, Charles Ollion, and Eric Moulines. Diffusion bridges vector quantized variational autoencoders. *International Conference on Machine Learning*, 2022.

[12] Daniel D Johnson, Jacob Austin, Rianne van den Berg, and Daniel Tarlow. Beyond in-place corruption: Insertion and deletion in denoising probabilistic models. *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models (INNF+)*, 2021.

[13] Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[14] Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021.

[15] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.

[16] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *International Conference on Learning Representations*, 2022.

[17] Hyungjin Chung, Byeongsu Sim, and Jong Chul Ye. Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[18] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped Langevin diffusion. *International Conference on Learning Representations*, 2022.

[19] Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. Diffusion schrödinger bridge with applications to score-based generative modeling. *Advances in Neural Information Processing Systems*, 2021.

[20] Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[21] Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[22] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[23] Darren J Wilkinson. *Stochastic Modelling for Systems Biology*. Chapman and Hall/CRC, 2018.

[24] David Levin, Yuval Peres, and Elizabeth Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[25] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *International Conference on Learning Representations*, 2021.

[26] Nikolay Savinov, Junyoung Chung, Mikolaj Binkowski, Erich Elsen, and Aaron van den Oord. Step-unrolled denoising autoencoders for text generation. *International Conference on Learning Representations*, 2022.

[27] Anirudh Goyal, Nan Rosemary Ke, Surya Ganguli, and Yoshua Bengio. Variational walkback: Learning a transition operator as a stochastic recurrent net. *Advances in Neural Information Processing Systems*, 2017.

[28] Min Jin Chong and David Forsyth. Effectively unbiased fid and inception score and where to find them. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[29] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University, 2016.

[30] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. *AAAI Conference on Artificial Intelligence*, 2018.

[31] Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. *International Society for Music Information Retrieval*, 2021.

[32] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-dpm: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. *International Conference on Learning Representations*, 2022.

[33] David F Anderson. A modified next reaction method for simulating chemical systems with time dependent propensities and delays. *The Journal of Chemical Physics*, 127(21):214107, 2007.

[34] Chie Furusawa, Shinya Kitaoka, Michael Li, and Yuri Odagiri. Generative probabilistic image colorization. *arXiv preprint arXiv:2109.14518*, 2021.

[35] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. *AAAI Conference on Artificial Intelligence*, 2018.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical Image Computing and Computer-assisted Intervention*, 2015.

[38] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *International Conference on Learning Representations*, 2017.

[39] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. *International Conference on Machine Learning*, 2018.

[40] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[41] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.2.1.

Appendix

Contents

The appendix is organized as follows. In Section A, we provide a short introduction to Continuous Time Markov Chains, including the relevant results we use in this work. Proofs for all the Propositions and Theorems from the main text are in Section B. We then describe in Section C some additional intuitions and forms of our proposed objective, $\mathcal{L}_{\text{CT}}$. In Section D, we describe how an additional direct denoising model supervision term can be added to the objective to improve empirical performance. Details for how we define the forward process in our model can be found in Section E. Section F describes in more detail how CTMCs can be simulated and includes the algorithmic description of tau-leaping. We argue in Section G that operating in discrete time forces an implicit assumption when using a factorized forward process on multi-dimensional data. Full experimental details for all investigations can be found in Section H as well as additional plots and results from our models. Finally, in Section I, we consider the social impacts of our research.

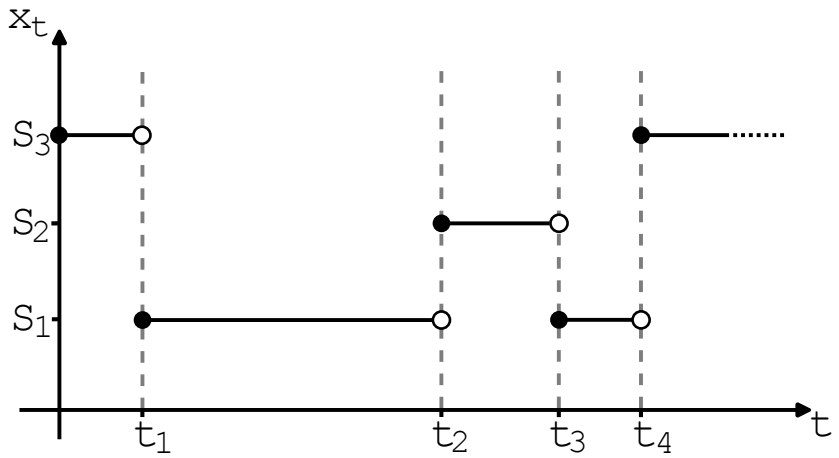


Figure 6: Schematic representation of a 1-dimensional CTMC with 3 states.

A Primer on Continuous Time Markov Chains

A Continuous Time Markov Chain (CTMC) is a right continuous stochastic process $\{x_t\}_{t\in[0,T]}$ satisfying the Markov property, with x_t taking values in a discrete state space $\mathcal{X}$. Since the CTMC is Markov, future behaviour of the process depends only on the current state and not the history. A schematic representation of a CTMC path is shown in Figure 6. The process repeatedly transitions from one state to another after having waited in the previous state for a randomly determined amount of time.

A CTMC can be completely characterised by its jumps and holding times. Specifically, the time between each jump or *holding time* is exponentially distributed with mean $\nu(x)$ where x is the state in which the process is holding. The next state that is jumped to is drawn from a jump probability distribution $r(\tilde{x}|x)$. The holding and jumping procedure is then repeated.

There is an equivalent definition involving the transition rate matrix, $R \in \mathbb{R}^{S\times S}$, that we use in the main paper. The transition rate matrix is defined as

$$R(\tilde{x}, x) = \lim_{\Delta t\to 0} \frac{q_{t|t-\Delta t}(x|\tilde{x}) - \delta_{x,\tilde{x}}}{\Delta t} \tag{3}$$

where $R(\tilde{x}, x)$ is the $(\tilde{x}, x)$ element of the transition rate matrix and $q_{t|t-\Delta t}(x|\tilde{x})$ is the infinitesimal transition probability of being in state x at time t given that the process was in state $\tilde{x}$ at time $t - \Delta t$. Conversely, the CTMC can itself be defined through this infinitesimal transition probability

$$q_{t|t-\Delta t}(x|\tilde{x}) = \delta_{x,\tilde{x}} + R(\tilde{x}, x)\Delta t + o(\Delta t) \tag{4}$$

where $o(\Delta t)$ represents terms that tend to zero at a faster rate than Δt. From this definition of the transition rate matrix, we can infer the following properties:

$$R(\tilde{x}, x) \geq 0 \quad \text{for} \quad \tilde{x} \neq x, \qquad R(x, x) \leq 0, \qquad R(x, x) = -\sum_{x'\neq x} R(x, x') \tag{5}$$

$R(\tilde{x}, x)$ is the rate at which probability mass moves from state $\tilde{x}$ to x. $R(x, x)$ is the total rate at which probability mass moves out of state x and is thus negative.

In the time-homogeneous case, R has simple relations to the jump and holding time definitions.

$$\nu(x) = -\frac{1}{R(x, x)} \qquad r(\tilde{x}|x) = (1 - \delta_{\tilde{x},x})\frac{R(x, \tilde{x})}{-R(x, x)}$$

In the time-inhomogeneous case, our transition rate matrix will now depend on time, R_t, and these simple relations to the jump and holding time definition do not hold. However, R_t will still follow equations (3), (4) and (5).

The CTMC transition probabilities satisfy the Kolmogorov forward and backward equations. For $t > s$,

$$\text{Kolmogorov forward equation} \quad \partial_t q_{t|s}(x|\tilde{x}) = \sum_y q_{t|s}(y|\tilde{x}) R_t(y, x)$$

$$\text{Kolmogorov backward equation} \quad \partial_s q_{t|s}(x|\tilde{x}) = -\sum_y R_s(\tilde{x}, y) q_{t|s}(x|y)$$

The Kolmogorov forward equation also gives us a differential equation for the marginals of the CTMC.

$$\partial_t q_t(x) = \sum_y q_t(y) R_t(y, x).$$

Exponential and Poisson Random Variables In the time homogeneous case, holding times are exponentially distributed with mean $\nu(x) = -1/R(x, x)$. The tau-leaping algorithm relies on the fact that the number of events in interval $[0, t]$ is Poisson distributed with mean $\frac{1}{\nu}t$ when the inter-event times are exponentially distributed with mean ν.

B Proofs

B.1 Proof of Proposition 1

Proof. We recall that a process $\{x_t\}_{t\in[0,T]}$ taking values in $\mathcal{X}$ is called a CTMC if it is right-continuous and satisfies the Markov property. Denote $\{y_t\}_{t\in[0,T]} = \{x_{T-t}\}_{t\in[0,T]}$ except at the jump times of the forward process τ_n with $n \in \mathbb{N}$, where $y_{T-\tau_n} = x_{\tau_n}^- = \lim_{t\leq\tau_n, t\to\tau_n} x_t$. Hence, $\{y_t\}_{t\in[0,T]}$ is almost surely equal to $\{x_{T-t}\}_{t\in[0,T]}$ and is right-continuous. Since the Markov property is symmetric, we get that $\{y_t\}_{t\in[0,T]}$ is a CTMC. We now compute its transition matrix. Let $x, \tilde{x} \in \mathcal{X}$ with $x \neq \tilde{x}$, using the Kolmogorov forward equation, we have

$$\partial_t p_{t|s}(\tilde{x}|x) = \textstyle\sum_{y\in\mathcal{X}} p_{t|s}(y|x)\hat{R}_{T-t}(y, \tilde{x}) \ ,$$

where $\{p_{t|s},\ s, t \in [0, T],\ t > s\}$ is the transition probability system associated with $\{y_t\}_{t\in[0,T]}$ and $\{\hat{R}_{T-t}\}_{t\in[0,T]}$ is the transition rate matrix associated with $\{y_t\}_{t\in[0,T]}$. Note that

$$\begin{aligned}
p_{t|s}(x = j|\tilde{x} = i) &= \mathbb{P}(y_t = j \mid y_s = i) \\
&= \mathbb{P}(x_{T-t} = j \mid x_{T-s} = i) \\
&= \mathbb{P}(x_{T-s} = i|x_{T-t} = j)\frac{\mathbb{P}(x_{T-t} = j)}{\mathbb{P}(x_{T-s} = i)} \\
&= q_{T-s|T-t}(\tilde{x} = i|x = j)\frac{q_{T-t}(x = j)}{q_{T-s}(\tilde{x} = i)}
\end{aligned}$$

where $\{q_{t|s}, s, t \in [0, T], t > s\}$ is the transition probability system associated with $\{x_t\}_{t\in[0,T]}$ and $\{q_t, t \in [0, T]\}$ are the marginals of $\{x_t\}_{t\in[0,T]}$. Now, writing the backward Kolmogorov equation for $\{x_t\}_{t\in[0,T]}$

$$\partial_s q_{t|s}(\tilde{x}|x) = -\textstyle\sum_{y\in\mathcal{X}} R_s(x, y) q_{t|s}(\tilde{x}|y)$$

Re-labeling the time indices we obtain,

$$\begin{aligned}
\partial_{T-t} q_{T-s|T-t}(\tilde{x}|x) &= -\textstyle\sum_{y\in\mathcal{X}} R_{T-t}(x, y) q_{T-s|T-t}(\tilde{x}|y) \\
\partial_t q_{T-s|T-t}(\tilde{x}|x) &= \textstyle\sum_{y\in\mathcal{X}} R_{T-t}(x, y) q_{T-s|T-t}(\tilde{x}|y)
\end{aligned}$$

Letting $s \to t$ and using that $\lim_{s\to t} q_{T-s|T-t}(x|\tilde{x}) = 0$, we get that

$$\begin{aligned}
\hat{R}_{T-t}(x,\tilde{x}) &= \lim_{s\to t} \partial_t p_{t|s}(\tilde{x}|x) \\
&= \lim_{s\to t} \partial_t \left(q_{T-s|T-t}(x|\tilde{x}) \frac{q_{T-t}(\tilde{x})}{q_{T-s}(x)} \right) \\
&= \lim_{s\to t} \left[\partial_t \left(q_{T-s|T-t}(x|\tilde{x}) \right) \frac{q_{T-t}(\tilde{x})}{q_{T-s}(x)} + q_{T-s|T-t}(x|\tilde{x}) \frac{\partial_t q_{T-t}(\tilde{x})}{q_{T-s}(x)} \right] \\
&= \lim_{s\to t} \partial_t \left(q_{T-s|T-t}(x|\tilde{x}) \right) \frac{q_{T-t}(\tilde{x})}{q_{T-s}(x)} \\
&= R_{T-t}(\tilde{x},x) \frac{q_{T-t}(\tilde{x})}{q_{T-t}(x)}
\end{aligned}$$

Re-labeling the time-indices on the rate matrices, we obtain

$$\hat{R}_t(x,\tilde{x}) = R_t(\tilde{x},x) \frac{q_t(\tilde{x})}{q_t(x)}$$

Now we write the marginal ratio $\frac{q_t(\tilde{x})}{q_t(x)}$ in a different form

$$\begin{aligned}
\frac{q_t(\tilde{x})}{q_t(x)} &= \sum_{x_0} \frac{p_{\text{data}}(x_0)}{q_t(x)} q_{t|0}(\tilde{x}|x_0) \\
&= \sum_{x_0} \frac{q_{0|t}(x_0|x)}{q_{t|0}(x|x_0)} q_{t|0}(\tilde{x}|x_0).
\end{aligned}$$

Substituting in this form for the marginal ratio concludes the proof.

□

B.2 Proof of Proposition 2

In this section, we detail two proofs for Proposition 2. The first is a formal proof using results from stochastic processes. We then provide a second informal proof for the same result to gain intuition into the $\mathcal{L}_{\text{CT}}$ objective that only relies on elementary results from CTMCs.

Proof 1 - Stochastic Processes

Proof. Let us write $\mathbb{Q}$ for the path measure of the forward CTMC with rate matrix R_t, $\hat{\mathbb{Q}}$ for the path measure of its exact time reversal and $\mathbb{P}^\theta$ for the path measure of the approximate reverse process with rate matrix $\hat{R}^\theta_t$. Also, we use superscripts to notate conditioning on the starting point, for example $\mathbb{Q}^{x_0}$ denotes the path measure of the forward process conditioned to start in x_0.

With this notation, we have

$$\begin{aligned}
-\log p_0^\theta(x_0) &= -\log \int p_{\text{ref}}(\mathrm{d}x_T) \int_{\{\hat{W}_T = x_0\}} \mathbb{P}^{\theta,x_T}(\mathrm{d}w) \\
&= -\log \int q_{T|0}(\mathrm{d}x_T) \int_{\{\hat{W}_T = x_0\}} \hat{\mathbb{Q}}^{x_T}(\mathrm{d}w) \frac{\mathrm{d}p_{\text{ref}}}{\mathrm{d}q_{T|0}}(x_T) \frac{\mathrm{d}\mathbb{P}^{\theta,x_T}}{\mathrm{d}\hat{\mathbb{Q}}^{x_T}}(w) \\
&= -\log \int q_{T|0}(\mathrm{d}x_T) \int \hat{\mathbb{Q}}(\mathrm{d}w|\hat{W}_0 = x_T, \hat{W}_T = x_0) \frac{\mathrm{d}p_{\text{ref}}}{\mathrm{d}q_{T|0}}(x_T) \frac{\mathrm{d}\mathbb{P}^{\theta,x_T}}{\mathrm{d}\hat{\mathbb{Q}}^{x_T}}(w) \mathbb{Q}^{x_T}\{\hat{W}_0 = x_0\} \\
&\leq \int q_{T|0}(\mathrm{d}x_T) \int \hat{\mathbb{Q}}(\mathrm{d}w|\hat{W}_0 = x_T, \hat{W}_T = x_0) \left\{ -\log \frac{\mathrm{d}\mathbb{P}^{\theta,x_T}}{\mathrm{d}\hat{\mathbb{Q}}^{x_T}}(w) \right\} + C,
\end{aligned}$$

where $\mathbb{P}^\theta, \hat{\mathbb{Q}}$ run in the reverse time direction. Writing $\hat{W}_s$ for a reverse path and integrating wrt $p_{\text{data}}(\mathrm{d}x_0)$ we have

$$\int p_{\text{data}}(x_0)[-\log p_0^\theta(x_0)] \leq \int p_{\text{data}}(x_0) \int q_{T|0}(\mathrm{d}x_T) \int \hat{\mathbb{Q}}(\mathrm{d}\hat{W}|\hat{W}_0 = x_T, \hat{W}_T = x_0)$$
$$\times \left\{ \int_{s=0}^{T} \hat{R}^\theta_{T-s}(\hat{W}_s)\mathrm{d}s - \sum_{s:\hat{W}_{s-}\neq\hat{W}_s} \log \mathbb{P}^\theta_{T-s}(\hat{W}_s|\hat{W}_{s-}) R^\theta_{T-s}(\hat{W}_{s-}) \right\} + C,$$

where $\hat{R}^\theta_t(x)$ is shorthand for $-\hat{R}^\theta_t(x,x)$.

When $x_0 \sim p_{\text{data}}, x_T \sim q_{T|0}(\cdot|x_0), \hat{W} \sim \hat{\mathbb{Q}}(\mathrm{d}W|\hat{W}_0 = x_T, \hat{W}_T = x_0)$, the reverse path is distributed according to $p_{\text{data}}(\mathrm{d}x_0)\mathbb{Q}_{x_0}(\mathrm{d}W)$ and therefore $(\hat{W}_{s-}, \hat{W}_s)$ is distributed like $(W_{T-s}, W_{(T-s)-})$ and thus we have

$$\int p_{\text{data}}(x_0)[-\log p_0^\theta(x_0)]$$
$$\leq \int p_{\text{data}}(x_0)\mathbb{Q}_{x_0}(\mathrm{d}W) \left\{ \int_{s=0}^{T} \hat{R}^\theta_{T-s}(W_{(T-s)-})\mathrm{d}s - \sum_{s:W_{(T-s)-}\neq W_{T-s}} \log \mathbb{P}^\theta_{T-s}(W_{(T-s)-}|W_{T-s}) \hat{R}^\theta_{T-s}(W_{T-s}) \right\} + C$$

Using Dynkin's lemma and the fact that $\mathbb{P}^\theta_t(x|y)\hat{R}^\theta_t(y) = \hat{R}^\theta_t(y,x)$ we can re-expresss this final line as

$$= \iint_{s=0}^{T} q_{T-s}(\mathrm{d}x) \left\{ \sum_{z\neq x} \hat{R}^\theta_{T-s}(x,z) - \sum_{z\neq x} R_{T-s}(x,z) \frac{\sum_{y\neq x} R_{T-s}(x,y)}{\sum_{z\neq x} R_{T-s}(x,z)} \log \hat{R}^\theta_{T-s}(y,x) \right\}$$
$$= \iint_{s=0}^{T} q_{T-s}(\mathrm{d}x) r_{T-s}(\mathrm{d}y|x) \left\{ \sum_{z\neq x} \hat{R}^\theta_{T-s}(x,z) - \sum_{z\neq x} R_{T-s}(x,z) \log \hat{R}^\theta_{T-s}(y,x) \right\}$$
$$= \iint_{s=0}^{T} q_s(\mathrm{d}x) r_s(\mathrm{d}y|x) \left\{ \sum_{z\neq x} \hat{R}^\theta_s(x,z) - \sum_{z\neq x} R_s(x,z) \log \hat{R}^\theta_s(y,x) \right\}$$

which rearranges to give the continuous time ELBO in the form of Proposition 2.

□

Proof 2 - Limit of Discrete Time ELBO

Proof. Consider a partitioning of $[0,T]$, $0 = t_0 < t_1 < \cdots < t_{k-1} < t_k < t_{k+1} < \cdots < t_{K-1} < t_K = T$. Let $t_k - t_{k-1} = \Delta t$ for all k. In subscripts we use k as a shorthand for t_k when this does not cause confusion. Considering a CTMC with this time partitioning converts the problem into a discrete time Markov Chain with forward transition kernel, $q_{k+1|k}(x_{k+1}|x_k)$ and parameterized reverse kernel, $p^\theta_{k|k+1}(x_k|x_{k+1})$. Therefore, we can write the negative ELBO in its discrete time form, $\mathcal{L}_{\text{DT}}$

$$\mathcal{L}_{\text{DT}}(\theta) = \mathbb{E}_{p_{\text{data}}(x_0)} \Big[\mathrm{KL}(q_{K|0}(x_K|x_0)||p_{\text{ref}}(x_K)) - \mathbb{E}_{q_{1|0}(x_1|x_0)} \left[\log p^\theta_{0|1}(x_0|x_1) \right]$$
$$+ \sum_{k=1}^{K-1} \mathbb{E}_{q_{k+1|0}(x_{k+1}|x_0)} \left[\mathrm{KL}(q_{k|k+1,0}(x_k|x_{k+1}, x_0)||p^\theta_{k|k+1}(x_k|x_{k+1})) \right] \Big]$$

In the following, we will write the transition kernels in terms of the CTMC rate matrices and take the limit as $\Delta t \to 0$ to obtain a continuous time negative ELBO.

First, consider one item from the inner sum of $\mathcal{L}_{\text{DT}}$

$$L_k = \mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)} \left[\mathrm{KL}(q_{k|k+1,0}(x_k|x_{k+1}, x_0)||p^\theta_{k|k+1}(x_k|x_{k+1})) \right]$$
$$= -\mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)q_{k|k+1,0}(x_k|x_{k+1},x_0)} \left[\log p^\theta_{k|k+1}(x_k|x_{k+1}) \right] + C$$
$$= -\mathbb{E}_{q_k(x_k)q_{k+1|k}(x_{k+1}|x_k)} \left[\log p^\theta_{k|k+1}(x_k|x_{k+1}) \right] + C$$

where we have absorbed terms that do not depend on θ into C. We now write $p^\theta_{k|k+1}(x_k|x_{k+1})$ in terms of $\hat{R}^\theta_k$.

$$p^\theta_{k|k+1}(x_k|x_{k+1}) = \delta_{x_k,x_{k+1}} + \hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)$$

$$\begin{aligned}
\log p^\theta_{k|k+1}(x_k|x_{k+1}) =& \log\left(\delta_{x_k,x_{k+1}} + \hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right) \\
=& \delta_{x_k,x_{k+1}} \log\left(1 + \hat{R}^\theta_k(x_k,x_k)\Delta t + o(\Delta t)\right) \\
&+ (1-\delta_{x_k,x_{k+1}}) \log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right) \\
=& \delta_{x_k,x_{k+1}} \left(\hat{R}^\theta_k(x_k,x_k)\Delta t + o(\Delta t)\right) \\
&+ (1-\delta_{x_k,x_{k+1}}) \log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right) \qquad (6)
\end{aligned}$$

where on the last line we have used the series expansion for $\log(1+z) = z - \frac{z^2}{2} + o(z^2)$ valid for $|z| \leq 1, z \neq -1$. For any finite $R^\theta_k(x_k,x_k)$, Δt can be taken small enough such that the series expansion holds. We now substitute this form for $\log p^\theta_{k|k+1}$ into L_k and further write the expectation over $q_{k+1|k}(x_{k+1}|x_k) = \delta_{x_k,x_{k+1}} + R_k(x_k,x_{k+1})\Delta t + o(\Delta t)$ as an explicit sum.

$$\begin{aligned}
L_k = -\mathbb{E}_{q_k(x_k)}\Bigg[\sum_{x_{k+1}}\Bigg\{&\left[\delta_{x_k,x_{k+1}} + R_k(x_k,x_{k+1})\Delta t + o(\Delta t)\right]\times \\
&\Big[\delta_{x_k,x_{k+1}}\left(\hat{R}^\theta_k(x_k,x_k)\Delta t + o(\Delta t)\right) \\
&+ \left(1-\delta_{x_k,x_{k+1}}\right)\log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right)\Big]\Bigg\}\Bigg] + C
\end{aligned}$$

$$\begin{aligned}
L_k = -\mathbb{E}_{q_k(x_k)}\Bigg[\sum_{x_{k+1}}\Bigg\{&\delta_{x_k,x_{k+1}}\hat{R}^\theta_k(x_k,x_k)\Delta t \\
&+ (1-\delta_{x_k,x_{k+1}})R_k(x_k,x_{k+1})\Delta t\times \\
&\log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right) + o(\Delta t)\Bigg\}\Bigg] + C
\end{aligned}$$

We can isolate $\hat{R}^\theta_k$ within the log through the following re-arrangement

$$\begin{aligned}
&\Delta t \log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right) \\
&= \Delta t \log \Delta t + \Delta t \log\left(\hat{R}^\theta_k(x_{k+1},x_k) + o(1)\right) \\
&= \Delta t \log \Delta t + \Delta t \log\left(1 + o(1)\right) + \Delta t \log\left(\hat{R}^\theta_k(x_{k+1},x_k)\right)
\end{aligned}$$

where the first two terms are independent of θ and tend to 0 as $\Delta t \to 0$. Note that we assume $\hat{R}^\theta_k(x_{k+1},x_k) > 0$ for $x_{k+1} \neq x_k$ pairs which have $R_k(x_k,x_{k+1}) > 0$. This assumption is valid because, for $x_{k+1} \neq x_k$, we have

$$\hat{R}^\theta_k(x_{k+1},x_k) = R_k(x_k,x_{k+1})\sum_{x_0}\frac{q_{k|0}(x_k|x_0)}{q_{k|0}(x_{k+1}|x_0)}p^\theta_{0|k}(x_0|x_{k+1})$$

and we assume $p^\theta_{0|k}(x_0|x_{k+1}) > 0$ which is valid when we parameterize $p^\theta_{0|k}$ with a softmax output. We assume an irreducible Markov chain, hence $q_{k|0} > 0$ for $t_k > 0$.

With this re-arrangement, and absorbing constant terms into C, we obtain

$$L_k = -\mathbb{E}_{q_k(x_k)}\Bigg[\sum_{x_{k+1}}\Bigg\{\delta_{x_k,x_{k+1}}\hat{R}_k^\theta(x_k,x_k)\Delta t \\ + (1-\delta_{x_k,x_{k+1}})R_k(x_k,x_{k+1})\Delta t\log\left(\hat{R}_k^\theta(x_{k+1},x_k)\right) \\ + o(\Delta t)\Bigg\}\Bigg] + C$$

$$L_k = -\mathbb{E}_{q_k(x_k)}\left[\hat{R}_k^\theta(x_k,x_k)\Delta t + \sum_{x_{k+1}\neq x_k} R_k(x_k,x_{k+1})\Delta t\log\hat{R}_k^\theta(x_{k+1},x_k) + o(\Delta t)\right]$$

The second term can be re-written so that it is more efficient to approximate with Monte Carlo. Currently the denoising model $p_{0|k}^\theta$ has to be evaluated for each term in the sum $\sum_{x_{k+1}\neq x_k}$ which would require multiple forward passes of the neural network. We can instead create a new probability distribution to sample from as follows. Define

$$r_k(x_{k+1}|x_k) = (1-\delta_{x_k,x_{k+1}})\frac{R_k(x_k,x_{k+1})}{\mathcal{Z}^k(x_k)}$$

where

$$\mathcal{Z}^k(x_k) = \sum_{x'_{k+1}\neq x_k} R_k(x_k,x'_{k+1})$$

So we now have

$$L_k = -\mathbb{E}_{q_k(x_k)r_k(x_{k+1}|x_k)}\left[\hat{R}_k^\theta(x_k,x_k)\Delta t + \mathcal{Z}^k(x_k)\Delta t\log\hat{R}_k^\theta(x_{k+1},x_k) + o(\Delta t)\right]$$

Examining the other terms in $\mathcal{L}_{\text{DT}}$ we have $\mathbb{E}_{p_{\text{data}}(x_0)}\left[\text{KL}(q_{K|0}(x_K|x_0)||p_{\text{ref}}(x_K))\right]$ which does not depend on θ and $\mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p_{0|1}^\theta(x_0|x_1)\right]$ which we expand here

$$\begin{aligned}
&\mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p_{0|1}^\theta(x_0|x_1)\right] \\
&\quad= \sum_{x_1}\{\delta_{x_1,x_0} + \Delta t R_1(x_0,x_1) + o(\Delta t)\}\log p_{0|1}^\theta(x_0|x_1) \\
&\quad= \log p_{0|1}^\theta(x_0|x_0) + \Delta t\sum_{x_1} R_1(x_0,x_1)\log p_{0|1}^\theta(x_0|x_1) + o(\Delta t) \\
&\quad= \Delta t\hat{R}_1^\theta(x_0,x_0) + \Delta t\sum_{x_1} R_1(x_0,x_1)\log p_{0|1}^\theta(x_0|x_1) + o(\Delta t)
\end{aligned}$$

where on the final line we have used eq 6. In summary,

$$\begin{aligned}
\mathcal{L}_{\text{DT}} =& \Delta t\mathbb{E}_{p_{\text{data}}(x_0)q_{1|0}(x_1|x_0)}\left[-\hat{R}_1^\theta(x_0,x_0) + \sum_{x_1} R_1(x_0,x_1)\log p_{0|1}^\theta(x_0|x_1)\right] \\
&- \Delta t\sum_{k=1}^{K-1}\mathbb{E}_{q_k(x_k)r_k(x_{k+1}|x_k)}\left[\hat{R}_k^\theta(x_k,x_k) + \mathcal{Z}^k(x_k)\log\hat{R}_k^\theta(x_{k+1},x_k)\right] \\
&+ o(\Delta t) + C
\end{aligned}$$

We now take the limit of $\mathcal{L}_{\text{DT}}$ as $\Delta t\to 0$ and $K\to\infty$.

$$\lim_{\Delta t\to 0}\mathcal{L}_{\text{DT}} = \mathcal{L}_{\text{CT}} = -\int_0^T \mathbb{E}_{q_t(x)r_t(\tilde{x}|x)}\left[\hat{R}_t^\theta(x,x) + \mathcal{Z}^t(x)\log\left(\hat{R}_t^\theta(\tilde{x},x)\right)\right]dt + C$$

We can estimate the integral with Monte Carlo if we consider it to be an expectation with respect to a uniform distribution over times $(0,T)$. We also write $\hat{R}_t^\theta(x,x)$ explicitly as the negative off diagonal row sum to obtain

$$\mathcal{L}_{\text{CT}}(\theta) = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(x)r_t(\tilde{x}|x)}\left[\left\{\textstyle\sum_{x'\neq x}\hat{R}_t^\theta(x,x')\right\} - \mathcal{Z}^t(x)\log\left(\hat{R}_t^\theta(\tilde{x},x)\right)\right] + C.$$

□

B.3 Proof of Proposition 3

Proof. We assume $q_{t|s}(\boldsymbol{x}_t^{1:D}|\boldsymbol{x}_s^{1:D})$ factorizes as $\prod_{d=1}^{D} q_{t|s}(x_t^d|x_s^d)$ where $q_{t|s}(x_t^d|x_s^d)$, $d = 1,\dots,D$ are the transition probabilities for independent singular dimensional CTMCs each with forward rate $R_t^d(\tilde{x}^d, x^d)$. In the following, we will drop time subscripts on x arguments. To find the correspondence between $R_t^{1:D}$ and R_t^d, we use the Kolmogorov forward equation

$$\partial_t q_{t|s}(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) = \sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{y}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) R_t^{1:D}(\boldsymbol{y}^{1:D}, \boldsymbol{x}^{1:D})$$

Substitute in our factorized form for $q_{t|s}$ into the LHS

$$\begin{aligned}
\partial_t q_{t|s}(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) &= \partial_t \left\{ \prod_{d=1}^{D} q_{t|s}(x^d|\tilde{x}^d) \right\} \\
&= \sum_{d=1}^{D} q_{t|s}(\boldsymbol{x}^{1:D\setminus d}|\tilde{\boldsymbol{x}}^{1:D\setminus d}) \partial_t q_{t|s}(x^d|\tilde{x}^d) \\
&= \sum_{d=1}^{D} q_{t|s}(\boldsymbol{x}^{1:D\setminus d}|\tilde{\boldsymbol{x}}^{1:D\setminus d}) \sum_{y^d} q_{t|s}(y^d|\tilde{x}^d) R_t^d(y^d, x^d) \\
&= \sum_{d=1}^{D} \sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{x}^{1:D\setminus d}|\tilde{\boldsymbol{x}}^{1:D\setminus d}) q_{t|s}(y^d|\tilde{x}^d) R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \boldsymbol{y}^{1:D\setminus d}} \\
&= \sum_{d=1}^{D} \sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{y}^{1:D\setminus d}|\tilde{\boldsymbol{x}}^{1:D\setminus d}) q_{t|s}(y^d|\tilde{x}^d) R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \boldsymbol{y}^{1:D\setminus d}} \\
&= \sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{y}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) \sum_{d=1}^{D} R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \boldsymbol{y}^{1:D\setminus d}}
\end{aligned}$$

We therefore obtain

$$\sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{y}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) R_t^{1:D}(\boldsymbol{y}^{1:D}, \boldsymbol{x}^{1:D}) = \sum_{\boldsymbol{y}^{1:D}} q_{t|s}(\boldsymbol{y}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) \sum_{d=1}^{D} R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \boldsymbol{y}^{1:D\setminus d}}$$

This must be true for all possible factorizable forward process transitions, $q_{t|s}$, including $q_{t|s}(\boldsymbol{y}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}) = \delta_{\boldsymbol{y}^{1:D}, \tilde{\boldsymbol{x}}^{1:D}}$. This choice gives us our forward rate relation

$$R_t^{1:D}(\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}) = \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}}$$

Substituting this into our expression for the reverse rate from Proposition 1 we obtain

$$\begin{aligned}
\hat{R}_t^{1:D}(\boldsymbol{x}^{1:D}, \tilde{\boldsymbol{x}}^{1:D}) &= \sum_{\boldsymbol{x}_0^{1:D}} \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \frac{q_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})}{q_t(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} \\
&= \sum_{\boldsymbol{x}_0^{1:D}} \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)} q_{0|t}(\boldsymbol{x}_0^{1:D}|\boldsymbol{x}^{1:D}) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} \\
&= \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}^{1:D}) \frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)} \sum_{\boldsymbol{x}_0^{1:D\setminus d}} q_{0|t}(\boldsymbol{x}_0^{1:D\setminus d}|x_0^d, \boldsymbol{x}^{1:D}) \\
&= \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}^{1:D}) \frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)}
\end{aligned}$$

□

B.4 Proof of Proposition 4

Proof. By the Kolmogorov forward equation applied to the forwards process, we have

$$\partial_t q_t(x_t) = \sum_y R_t(y, x_t) q_t(y)$$

In addition, applying the Kolmogorov forward equation to the reverse process, which has the same marginals as the forward but time-reversed, we get

$$-\partial_t q_t(x_t) = \sum_y \hat{R}_t(y, x_t) q_t(y)$$

Summing these two equations gives

$$\sum_y \left\{ R_t(y, x_t) + \hat{R}_t(y, x_t) \right\} q_t(y) = 0$$

Therefore, by comparison with the Kolmogorov equation, $R_t + \hat{R}_t$ is the rate matrix of a CTMC with invariant distribution q_t. □

B.5 Proof of Theorem 1

In this section, we derive a bound on the error of our tau-leaping diffusion model. Because the tau-leaping approximation is only interesting in the case where multiple jumps are made along different dimensions in a single step, we choose to make the dependence of our bound on the dimension of our model explicit, rather than simply considering the case of fixed D and $\tau \to 0$.

Recall from the main text that we have a time-homogeneous rate matrix R_t on $\mathcal{X}$, from which we construct the factorised rate matrix $R_t^{1:D}$ on $\mathcal{X}^D$ by setting $R_t^d = R_t$ for each d, and will denote $|R| = \sup_{t \in [0,T], x \in \mathcal{X}} |R_t(x, x)|$, and let t_{mix} be the (1/4)-mixing time of the CTMC with rate R_t. We also define addition on the state space $\mathcal{X}^D$ using a mapping from $\mathcal{X}$ to $\mathbb{Z}$ as in Section 4.3 and component-wise addition.

Theorem 1. *For any $D \geq 1$ and distribution p_{data} on $\mathcal{X}^D$, let $\{x_t\}_{t \in [0,T]}$ be a CTMC starting in p_{data} with rate matrix $R_t^{1:D}$ as above. Suppose that $\hat{R}_t^{\theta\, 1:D}$ is an approximation to the reverse rate matrix and let $(y_k)_{k=0,1,\ldots,N}$ be a tau-leaping approximation to the reverse dynamics with maximum step size τ. Suppose further that there is some constant $M > 0$ independent of D such that*

$$\sum_{y \neq x} \left| \hat{R}_t^{1:D}(x, y) - \hat{R}_t^{\theta\, 1:D}(x, y) \right| \leq M$$

for all $t \in [0, T]$. Then under the assumptions listed below, there are constants $C_1, C_2 > 0$ depending on $\mathcal{X}$ and R_t but not D such that, if $\mathcal{L}(y_0)$ denotes the law of y_0, we have the total variation bound

$$||\mathcal{L}(y_0) - p_{\text{data}}||_{\text{TV}} \leq 3MT + \left\{ \left(|R| S D C_1 \right)^2 + \tfrac{1}{2} C_2 (M + C_1 S D |R|) \right\} \tau T + 2 \exp\left\{ - \frac{T \log^2 2}{t_{\text{mix}} \log 4D} \right\}$$

The above theorem holds under the following assumptions, where we write $x \sim y$ for $x, y \in S^D$ if they differ in at most one coordinate.

Assumption 1. *The data distribution p_{data} is strictly positive.*

Assumption 2. *There exists a constant $C_1 > 0$, depending on S and R_t but not D, such that for all $t \in [0, T]$ and $x, y \in S^D$ such that $x \sim y$, we have*

$$\frac{q_t(x)}{q_t(y)} \leq C_1.$$

Assumption 3. *There exists a constant $C_2 > 0$, depending on S and R_t but not D, such that for all $t \in [0, T]$ and all $x, y \in S^D$ such that $x \sim y$, we have*

$$\sum_z \left| \hat{R}_t(x, x + z) - \hat{R}_t(y, y + z) \right| \leq C_2.$$

If instead we were to allow C_1 and C_2 to depend on the dimension D, then Assumptions 2 and 3 follow trivially from Assumption 1 and the finiteness of the state space. However, we choose the stronger formulation above in order to make explicit the dependence of the error bound on the dimension, as previously explained.

As remarked in the main text, in most cases of practical interest (including the two examples explored in Section 6), Assumption 3 holds only approximately. However, we still expect the bound in Assumption 3 to hold whenever x, y are in addition chosen such that the tau-leaping approximation of the reverse process makes a jump between them with reasonably high probability. For example, in the case where our data is ordinal, we expect that for any $x \sim y$ jumps from x to y are only common when x is close to y, and thus $\hat{R}_t(x, x+z)$ and $\hat{R}_t(y, y+z)$ should be reasonably close whenever a jump from x to y occurs. Under a weaker assumption of this form, the proof of Theorem 1 can be adapted to work along similar lines, at the cost of a significant increase in technicality. We therefore choose to focus on the simpler case where Assumption 3 holds as it illustrates the key ideas.

In order to prove Theorem 1 we will require the following lemmas.

Proposition 5. *Let $(x_t)_{t\in[0,T]}$ and $(y_t)_{t\in[0,T]}$ be continuous time Markov chains on a finite state space S with generators G_t and H_t respectively which are both bounded and continuous in t. Let the Markov kernels associated to X and Y be K and L respectively. Then for any probability distribution ν on S we have*

$$||\nu K - \nu L||_{\text{TV}} \leq \int_0^T \sup_{x\in S} \Big\{ \sum_{y\neq x} |G_t(x,y) - H_t(x,y)| \Big\} \, \mathrm{d}t$$

Proof. We define a coupling of $(x_t)_{t\in[0,T]}$ and $(y_t)_{t\in[0,T]}$ as follows, based on the construction in Chapter 20.1 of [24]. First take $Z \sim \nu$ and set $x_0 = y_0 = Z$. Also define the variables $\tilde{x}_0 = \tilde{y}_0 = Z$.

Next, fix λ such that $|G_t(x,x)|, |H_t(x,x)| \leq \lambda$ for all $x \in S, t \in [0,T]$, let $(N_s)_{1\leq s\leq T}$ be a Poisson process on $[0,T]$ of rate λ, and set $N_0 = 0$. We write $N = N_T$, and $S_1, S_2, \ldots, S_N$ for the arrival times and set $S_{n+1} = T$. We construct x_t and y_t for $t > 0$ inductively as follows. For $t \in [0, S_1)$ let $x_t = y_t = x_0$. Let $1 \leq j \leq N$. Given $(x_r : r < S_j)$, $(y_r : r < S_j)$, and $\tilde{x}_j, \tilde{y}_j$, define the following probability measures

$$\rho_j(\tilde{x}_j, w) := \begin{cases} G_{S_j}(\tilde{x}_j, w)/\lambda, & w \neq \tilde{x}_j \\ 1 - G_{S_j}(\tilde{x}_j, w)/\lambda, & w = \tilde{x}_j, \end{cases}$$
$$\rho'_j(\tilde{y}_j, w) := \begin{cases} H_{S_j}(\tilde{y}_j, w)/\lambda, & w \neq \tilde{y}_j \\ 1 - H_{S_j}(\tilde{y}_j, w)/\lambda, & w = \tilde{y}_j. \end{cases}$$

Sample $(\tilde{x}_{j+1}, \tilde{y}_{j+1})$ from a maximal coupling of (ρ_j, ρ'_j) and for $t \in [S_j, S_{j+1})$ set $x_t = \tilde{x}_{j+1}$, $y_t = \tilde{y}_{j+1}$. Finally set $x_T = x_{S_N}$ and $y_T = y_{S_N}$.

Now, observe that $(x_t, y_t)_{t\in[0,T]}$ defined in this way is a coupling of the given Markov chains. Moreover,

$$\begin{aligned} ||\nu K - \nu L||_{\text{TV}} &\leq \mathbb{P}(x_T \neq y_T) \\ &= \mathbb{E}\left[\sum_{j=1}^{N} \mathbb{I}\{x_s = y_s,\, s < S_j\} \, \mathbb{I}\{x_{S_j} \neq y_{S_j}\} \right] \\ &= \sum_{n=0}^{\infty} \frac{\lambda^n e^{-\lambda}}{n!} \sum_{j=0}^{n} \mathbb{E}\left[\mathbb{I}\{x_s = y_s,\, s < S_j\} \, \mathbb{I}\{x_{S_j} \neq y_{S_j}\} \right] \end{aligned}$$

and using the fact that jumps are coupled maximally

$$
\begin{aligned}
&= \sum_{n=0}^{\infty} \frac{\lambda^n e^{-\lambda}}{n!} \sum_{j=0}^{n} \mathbb{E}\Big[\mathbb{I}\left\{x_s = y_s,\, s < S_j\right\} \times \|\rho_j(X_{S_{j-1}}, \cdot) - \tilde{\rho}_j(X_{S_{j-1}}, \cdot)\|_{\mathrm{TV}}\Big] \\
&= \sum_{n=0}^{\infty} \frac{\lambda^n e^{-\lambda}}{n!} \sum_{j=0}^{n} \mathbb{E}\Big[\mathbb{I}\left\{x_s = y_s,\, s < S_j\right\} \frac{1}{\lambda} \sum_z |G_{S_j}(x_{S_{j-1}}, z) - H_{S_j}(x_{S_{j-1}}, z)|\Big] \\
&= \frac{1}{\lambda} \mathbb{E}\Big[\sum_{s: x_s \neq x_{s-}} \sum_z |G_s(x_{s-}, z) - H_s(x_{s-}, z)|\Big] \\
&= \frac{1}{\lambda} \int_{s=0}^{T} \mathbb{E}\Big[\lambda \sum_z |G_s(x_{s-}, z) - H_s(x_{s-}, z)|\Big] \\
&= \int_{s=0}^{T} \mathbb{E}\Big[\sum_z |G_s(x_{s-}, z) - H_s(x_{s-}, z)|\Big] \mathrm{d}s
\end{aligned}
$$

as required. □

Proposition 6. *For all $t \in [0, T]$ and $x, y \in \mathcal{X}^D$ such that $x \sim y$, we have*

$$|\partial_t \hat{R}_t(x, y)| \leq 2|R|^2 SDC_1^2$$

Moreover, it follows that $\hat{R}_t$ is bounded and continuous in t.

Proof. Omitting the superscripts for brevity where the notation is clear, we have

$$
\begin{aligned}
\left|\partial_t \hat{R}_t^{1:D}(x^{1:D}, y^{1:D})\right| &= \left|R_t(y, x) \partial_t \left\{\frac{q_t(y)}{q_t(x)}\right\}\right| \\
&= \left|R_t(y, x) \left\{\frac{q_t(y)}{q_t(x)} \frac{\sum_z R_t(z, y) q_t(z)}{q_t(y)} - \frac{q_t(y)}{q_t(x)} \frac{\sum_z R_t(z, x) q_t(z)}{q_t(x)}\right\}\right| \\
&\leq 2|R|^2 SDC_1^2
\end{aligned}
$$

where the second line follows from Kolmogorov's forward equation and the final inequality follows from Assumption 2 plus the fact that $R_t(z, x)$ (resp. $R_t(z, y)$) is only non-zero when $x \sim z$ (resp. $y \sim z$), and there are at most $|S||D|$ values of x (resp. y) for which this holds. □

We now give the proof of Theorem 1.

Proof of Theorem 1. Let us label the time steps used in tau-leaping by $0 = t_0 < t_1 < \cdots < t_N = T$, denote $\tau_k = t_k - t_{k-1}$, and denote the target stationary distribution by $\pi^D(x^{1:D}) = \prod_{d=1}^{D} \pi(x^d)$, where π is the invariant distribution of the single-dimensional transition matrix R_t^1.

Also, let $\mathcal{R}_k^{\theta,(\tau)}$ be the Markov kernel corresponding to applying the tau-leaping approximation with rate matrix $\hat{R}_{t_k}^{\theta}$ to move from t_k to t_{k-1}, and denote $\mathcal{R}^{\theta,(\tau)} = \mathcal{R}_N^{\theta,(\tau)} \mathcal{R}_{N-1}^{\theta,(\tau)} \ldots \mathcal{R}_1^{\theta,(\tau)}$ so that $\mathcal{R}^{\theta,(\tau)}$ expresses the full dynamics of the tau-leaping process and we have $\mathcal{L}(\hat{y}_0) = \pi^D \mathcal{R}^{\theta,(\tau)}$.

Then, as in [19] we can decompose

$$||\pi^D \mathcal{R}^{\theta,(\tau)} - p_d||_{\mathrm{TV}} \leq ||\pi^D \mathcal{R}^{\theta,(\tau)} - \pi^D (\mathbb{P}^R)_{T|0}||_{\mathrm{TV}} + ||\pi^D - q_T||_{\mathrm{TV}}$$

where $\mathbb{P}^R$ is the path measure of the exact reverse process.

We deal with the second term first. Let t_{mix} be the (1/4)-mixing time of the single-dimension CTMC with rate matrix R_t^1, i.e.

$$t_{\mathrm{mix}} = \inf \left\{ t \geq 0 : \sup_{x_0^1 \in S} ||q_{t|0}(\,\cdot\,|x_0^1) - \pi||_{\mathrm{TV}} \leq \frac{1}{4} \right\}$$

It then follows from

$$||q_{t|0}(\,\cdot\,|x_0^{1:D}) - \pi^D||_{\text{TV}} \leq \sum_{d=1}^{D} ||q_{t|0}(\,\cdot\,|x_0^d) - \pi||_{\text{TV}}$$

that t^D_{mix}, the (1/4)-mixing time of the full CTMC with rate matrix $R_t^{1:D}$, satisfies the inequality $t^D_{mix} \leq \{1 + \lceil \log_2 D \rceil\} t_{\text{mix}}$. If we view $(x_{mt^D_{mix}})_{m\in\mathbb{N}}$ as a discrete-time Markov chain, then standard results on Markov chain mixing (see, for example, Chapter 4.5 of [24]) show that

$$||q_{mt^D_{mix}|0}(\,\cdot\,|x_0^{1:D}) - \pi^D||_{\text{TV}} \leq 2^{-m}$$

It then follows that for any $T \geq 0$ we have

$$||\pi^D - q_T||_{\text{TV}} \leq 2\exp\left\{-\frac{T\log 2}{t^D_{mix}}\right\} \leq 2\exp\left\{-\frac{T\log^2 2}{t_{\text{mix}}\log 4D}\right\}$$

completing the bound on the second term.

To bound the first term, we define $\mathcal{P}_k = (\mathbb{P}^R)_{T-t_{k-1}|T-t_k}$ and decompose it as

$$\begin{aligned}
||\pi\mathcal{R}^{\theta,(\tau)} - \pi(\mathbb{P}^R)_{T|0}||_{\text{TV}} &\leq \sup_\nu ||\nu\mathcal{R}_N^{\theta,(\tau)} \dots \mathcal{R}_1^{\theta,(\tau)} - \nu\mathcal{P}_N \dots \mathcal{P}_1||_{\text{TV}} \\
&\leq \sup_\nu ||\nu\mathcal{R}_N^{\theta,(\tau)}\mathcal{R}_{N-1}^{\theta,(\tau)} \dots \mathcal{R}_1^{\theta,(\tau)} - \nu\mathcal{R}_N^{\theta,(\tau)}\mathcal{P}_{N-1} \dots \mathcal{P}_1||_{\text{TV}} \\
&\quad + \sup_\nu ||\nu\mathcal{R}_N^{\theta,(\tau)}\mathcal{P}_{N-1} \dots \mathcal{P}_1 - \nu\mathcal{P}_N\mathcal{P}_{N-1} \dots \mathcal{P}_1||_{\text{TV}} \\
&\leq \sup_\nu ||\nu\mathcal{R}_{N-1}^{\theta,(\tau)} \dots \mathcal{R}_1^{\theta,(\tau)} - \nu\mathcal{P}_{N-1} \dots \mathcal{P}_1||_{\text{TV}} + \sup_\nu ||\nu\mathcal{R}_N^{\theta,(\tau)} - \nu\mathcal{P}_N||_{\text{TV}} \\
&\leq \sum_{k=1}^{N} \sup_\nu ||\nu\mathcal{R}_k^{\theta,(\tau)} - \nu\mathcal{P}_k||_{\text{TV}}
\end{aligned}$$

by proceeding inductively. So it suffices to find bounds on the total variation distance accumulated on each interval $[t_{k-1}, t_k]$.

Let $\mathcal{R}_k^\theta$ be the Markov kernel corresponding to running the chain from t_k to t_{k-1} with constant rate matrix $\hat{R}^\theta_{t_k}$. Since by Proposition 6 the reverse rate matrix $\hat{R}_t$ is bounded and continuous in t, using Proposition 5 we made deduce that for any distribution ν on S we have

$$\begin{aligned}
||\nu\mathcal{P}_k - \nu\mathcal{R}_k^\theta||_{\text{TV}} &\leq \int_{t_{k-1}}^{t_k} \sup_{x\in S}\Big\{\sum_{y\neq x} \big|\hat{R}_t(x,y) - \hat{R}^\theta_{t_k}(x,y)\big|\Big\}\,\mathrm{d}t \\
&\leq \int_{t_{k-1}}^{t_k} \sup_{x\in S}\Big\{\sum_{y\neq x} \big|\hat{R}_t(x,y) - \hat{R}_{t_k}(x,y)\big|\Big\}\,\mathrm{d}t \\
&\quad + \int_{t_{k-1}}^{t_k} \sup_{x\in S}\Big\{\sum_{y\neq x} \big|\hat{R}_{t_k}(x,y) - \hat{R}^\theta_{t_k}(x,y)\big|\Big\}\,\mathrm{d}t
\end{aligned}$$

The first half of this expression can be bounded using the Mean Value Theorem, according to

$$\begin{aligned}
\int_{t_{k-1}}^{t_k} \sup_{x\in S}\Big\{\sum_{y\neq x} \big|\hat{R}_t(x,y) - \hat{R}_{t_k}(x,y)\big|\Big\}\,\mathrm{d}t &\leq \int_{t_{k-1}}^{t_k} |t - t_k| \cdot 2|R|^2 S^2 D^2 C_1^2\,\mathrm{d}t \\
&\leq \big(|R|SDC_1\tau_k\big)^2
\end{aligned}$$

where in the first line we have used that the summand is only non-zero when $y \sim x$, and there are at most $|S||D|$ values of y for which this holds. The second term can be bounded using condition (2), to get

$$\int_{t_{k-1}}^{t_k} \sup_{x\in S}\Big\{\sum_{y\neq x} \big|\hat{R}_{t_k}(x,y) - \hat{R}^\theta_{t_k}(x,y)\big|\Big\}\,\mathrm{d}t \leq M\tau_k$$

Combining these two expressions, we get a bound on $||\nu\mathcal{P}_k - \nu\mathcal{R}_k^\theta||_{\text{TV}}$.

$$||\nu\mathcal{P}_k - \nu\mathcal{R}_k^\theta||_{\text{TV}} \leq \left(|R|SDC_1\tau_k\right)^2 + M\tau_k$$

It remains to bound $||\nu\mathcal{R}_k^\theta - \nu\mathcal{R}_k^{\theta,(\tau)}||_{\text{TV}}$. Note that performing tau-leaping with rate matrix $\hat{R}_{t_k}^\theta$ starting in x_{t_k} is equivalent to running a continuous time Markov chain from time t_k to t_{k-1} with constant rate matrix $\hat{R}_{t_k}^{\theta,(\tau)}$ given by

$$\hat{R}_{t_k}^{\theta,(\tau)}(x,y) = \hat{R}_{t_k}^\theta(x_{t_k}, y - x + x_{t_k})$$

(followed potentially by a clamping operation to keep us within $\mathcal{X}^D$). By an analogous argument to the proof of Proposition 5,

$$||\delta_{x_{t_k}}\mathcal{R}_k^\theta - \delta_{x_{t_k}}\mathcal{R}_k^{\theta,(\tau)}||_{\text{TV}} \leq \int_{t_{k-1}}^{t_k} \mathbb{E}\Big[\sum_{y\neq x_t} |\hat{R}_{t_k}^\theta(x_t,y) - \hat{R}_{t_k}^\theta(x_{t_k}, y - x_t + x_{t_k})|\Big]\,\mathrm{d}t$$

where the expectation is taken over $(x_t)_{t\in[t_{k-1},t_k]}$ distributed according to the exact CTMC with rate matrix $\hat{R}_{t_k}^\theta$. (Note we have disregarded the clamping operation, since this can only decrease the resulting total variation distance.)

We may rewrite this bound in terms of the exact reverse process using condition (2) to get

$$||\delta_{x_{t_k}}\mathcal{R}_k^\theta - \delta_{x_{t_k}}\mathcal{R}_k^{\theta,(\tau)}||_{\text{TV}} \leq \int_{t_{k-1}}^{t_k} \mathbb{E}\Big[2M + \sum_{y\neq x_t} |\hat{R}_{t_k}(x_t,y) - \hat{R}_{t_k}(x_{t_k}, y - x_t + x_{t_k})|\Big]\,\mathrm{d}t$$

Let J_t be the number of jumps that (x_t) makes between t_k and t, and label the times of these jumps as $s_1,\ldots,s_j$ where $t \leq s_1 \leq \cdots \leq s_j \leq t_k$ and $j = J_t$ for convenience. Then by Assumption 3, we have

$$\begin{aligned}\sum_{y\neq x_t} |\hat{R}_{t_k}(x_t,y) - \hat{R}_{t_k}(x_{t_k}, y - x_t + x_{t_k})| &\leq \sum_z |\hat{R}_{t_k}(x_t, x_t + z) - \hat{R}_{t_k}(x_{s_1}, x_{s_1} + z)| + \ldots \\ &\quad + \sum_z |\hat{R}_{t_k}(x_{s_j}, x_{s_j} + z) - \hat{R}_{t_k}(x_{t_k}, x_{t_k} + z)| \\ &\leq C_2 J_t\end{aligned}$$

where we have made the substitution $z = y - x_{t_k}$. We conclude that

$$\begin{aligned}||\delta_{x_{t_k}}\mathcal{R}_k^\theta - \delta_{x_{t_k}}\mathcal{R}_k^{\theta,(\tau)}||_{\text{TV}} &\leq \int_{t_{k-1}}^{t_k} \mathbb{E}\left[2M + C_2 J_t\right]\,\mathrm{d}t \\ &\leq 2M|t_k - t_{k-1}| + C_2 \int_{t_{k-1}}^{t_k} |t_k - t| \cdot \sup_x |\hat{R}_{t_k}^\theta(x,x)|\,\mathrm{d}t \\ &\leq 2M\gamma_k + \frac{1}{2}C_2|\hat{R}_{t_k}^\theta|\tau_k^2 \\ &\leq 2M\tau_k + \frac{1}{2}C_2(M + C_1SD|R|)\tau_k^2\end{aligned}$$

where to bound $\mathbb{E}[J_t]$ we have observed that jumps of (x_t) occur at a rate bounded above by $\sup_x |\hat{R}_{t_k}^\theta(x,x)|$, and in the last line we have used the condition (2) and Assumption 2. Since the above holds for any choice of x_{t_k}, it follows that

$$\sup_\nu ||\nu\mathcal{R}_k^\theta - \nu\mathcal{R}_k^{\theta,(\tau)}||_{\text{TV}} \leq 2M\tau_k + \frac{1}{2}C_2(M + C_1SD|R|)\tau_k^2$$

Summing over k and putting all our bounds together, we get

$$||\mathcal{L}(y_0) - p_{\text{data}}||_{\text{TV}} \leq 3MT + \left\{\left(|R|SDC_1\right)^2 + \tfrac{1}{2}C_2(M + C_1SD|R|)\right\}\tau T + 2\exp\left\{-\tfrac{T\log^2 2}{t_{\text{mix}}\log 4D}\right\}$$

as required. □

C Continuous Time ELBO Details

C.1 Comparison with the Discrete Time ELBO

It is easiest to gain intuition on the $\mathcal{L}_{\text{CT}}$ objective by comparing it to its discrete time counterpart, $\mathcal{L}_{\text{DT}}$, and examining the way in which $\mathcal{L}_{\text{DT}}$ in the limit becomes $\mathcal{L}_{\text{CT}}$ when we take the time step size to be very small. We repeat the definition of $\mathcal{L}_{\text{CT}}$ here for convenience

$$\mathcal{L}_{\text{CT}}(\theta) = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(x)r_t(\tilde{x}|x)}\left[\left\{\sum_{x'\neq x}\hat{R}_t^\theta(x,x')\right\} - \mathcal{Z}^t(x)\log\left(\hat{R}_t^\theta(\tilde{x},x)\right)\right] + C.$$

Recall that a single term from the KL sum in $\mathcal{L}_{\text{DT}}$ up to an additive constant independent of θ is

$$-\mathbb{E}_{q_k(x_k)q_{k+1|k}(x_{k+1}|x_k)}\left[\log p^\theta_{k|k+1}(x_k|x_{k+1})\right].$$

Minimizing this term is to sample (x_k, x_{k+1}) from the forward dynamics and then maximize the assigned model probability for the pairing in the reverse direction. A similar idea can be used to understand $\mathcal{L}_{\text{CT}}$. First, we write $\log p^\theta_{k|k+1}(x_k|x_{k+1})$ in terms of $\hat{R}^\theta_k$ as

$$\begin{aligned}\log p^\theta_{k|k+1}(x_k|x_{k+1}) =& \delta_{x_k,x_{k+1}}\left(\hat{R}^\theta_k(x_k,x_k)\Delta t + o(\Delta t)\right)\\ &+ (1-\delta_{x_k,x_{k+1}})\log\left(\hat{R}^\theta_k(x_{k+1},x_k)\Delta t + o(\Delta t)\right)\end{aligned}$$

where we have separated the cases when $x_k = x_{k+1}$ and when $x_k \neq x_{k+1}$ (see the proof of $\mathcal{L}_{\text{CT}}$ for the full details). The first term will become the $\sum_{x'\neq x}\hat{R}^\theta_t(x,x')$ term in $\mathcal{L}_{\text{CT}}$ whilst the second term will become the $\mathcal{Z}^t(x)\log\left(\hat{R}^\theta_t(\tilde{x},x)\right)$ term. Now, when we minimize $\mathcal{L}_{\text{CT}}$, we are sampling $(x,\tilde{x})$ from the forward process and then maximizing the assigned model probability for the pairing in the reverse direction, just as in $\mathcal{L}_{\text{DT}}$. The slight extra complexity comes from the fact we are considering the case when $x_k = x_{k+1}$ and the case when $x_k \neq x_{k+1}$ separately. When $x_k = x_{k+1}$, this corresponds to the first term in $\mathcal{L}_{\text{CT}}$ which we can see is minimizing the reverse rate out of x which is exactly maximizing the model probability for no transition to occur. When $x_k \neq x_{k+1}$, this corresponds to the second term in $\mathcal{L}_{\text{CT}}$, which is maximizing the reverse rate from $\tilde{x}$ to x which in turn maximizes the model probability for the $\tilde{x}$ to x transition to occur.

C.2 Conditional Form

For the conditional form of $\mathcal{L}_{\text{CT}}$, denoted as $\bar{\mathcal{L}}_{\text{CT}}$, we instead upper bound the negative conditional model log-likelihood, $\mathbb{E}_{p_{\text{data}}(x_0,y)}[-\log p^\theta_0(x_0|y)]$ where y is our conditioner. $\bar{\mathcal{L}}_{\text{CT}}$ has the following form

$$\bar{\mathcal{L}}_{\text{CT}}(\theta) = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)p_{\text{data}}(x_0,y)q_{t|0}(x|x_0)r_t(\tilde{x}|x)}\left[\left\{\sum_{x'\neq x}\hat{R}^\theta_t(x,x'|y)\right\} - \mathcal{Z}^t(x)\log\left(\hat{R}^\theta_t(\tilde{x},x|y)\right)\right] + C,$$

where

$$\begin{aligned}\hat{R}^\theta_t(x,\tilde{x}|y) =& R_t(\tilde{x},x)\sum_{x_0}\frac{q_{t|0}(\tilde{x}|x_0)}{q_{t|0}(x|x_0)}p^\theta_{0|t}(x_0|x,y) \quad \text{for} \quad x\neq\tilde{x}.\\ =& -\sum_{x'\neq x}\hat{R}^\theta_t(x,x'|y) \quad \text{for} \quad x=\tilde{x}\end{aligned}$$

This follows easily from considering the conditional form of the discret time ELBO, $\bar{\mathcal{L}}_{\text{DT}}$ and using the same arguments as before to go from discrete time to continuous time.

$$\mathbb{E}_{p_{\text{data}}(x_0,y)}[-\log p^\theta_0(x_0|y)] \leq \mathbb{E}_{p_{\text{data}}(x_0,y)q_{1:K|0}(x_{1:K}|x_0)}\left[-\log\frac{p^\theta_{0:K}(x_{0:K}|y)}{q_{1:K|0}(x_{1:K}|x_0)}\right] = \bar{\mathcal{L}}_{\text{DT}}$$

C.3 Continuous Time ELBO with Factorization Assumptions

In the following Proposition, we show the form of $\mathcal{L}_{\text{CT}}$ when we use a factorized forward process. We note that in the proof we rearrange the sampling distribution from $p_{\text{data}}(\boldsymbol{x}_0^{1:D})q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})r_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}^{1:D})$ to $p_{\text{data}}(\boldsymbol{x}_0^{1:D})\psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})\phi_t(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D},\boldsymbol{x}_0^{1:D})$. This is not strictly necessary but it allows us to analytically sum over the intermediate $\boldsymbol{x}^{1:D}$ variable which greatly reduces the variance of the resulting objective.

Proposition 7. *The $\mathcal{L}_{\text{CT}}$ objective when we substitute in the factorized forms for the forward and reverse process given in Proposition 3 is*

$$\mathcal{L}_{\text{CT}} = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)p_{\text{data}}(\boldsymbol{x}_0^{1:D})q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})}\left[\sum_{d=1}^{D}\sum_{x'^d\neq x^d}\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},x'^d)\right]$$
$$- T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)p_{\text{data}}(\boldsymbol{x}_0^{1:D})\psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})}\left[\sum_{d=1}^{D}\sum_{x^d\neq\tilde{x}^d}\phi_t(x^d|\tilde{\boldsymbol{x}}^{1:D},\boldsymbol{x}_0^{1:D})\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D/d}\circ x^d)\log\left(\hat{R}_t^{\theta\,d}(\tilde{\boldsymbol{x}}^{1:D},x^d)\right)\right]$$
$$+ C$$

with

$$\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},\tilde{x}^d) = R_t^d(\tilde{x}^d,x^d)\sum_{x_0^d}p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}^{1:D})\frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)}$$

$$\mathcal{Z}^t(\boldsymbol{x}^{1:D}) = \sum_{d=1}^{D}\sum_{\tilde{x}^d\neq x^d}R_t^d(x^d,\tilde{x}^d)$$

$$\phi_t(x^d|\tilde{\boldsymbol{x}}^{1:D},\boldsymbol{x}_0^{1:D}) = \frac{R_t^d(x^d,\tilde{x}^d)q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d}\circ x^d|\boldsymbol{x}_0^{1:D})}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d}\circ x^d)\sum_{d'=1}^{D}\sum_{x'^{d'}\neq\tilde{x}^{d'}}\frac{R_t^{d'}(x'^{d'},\tilde{x}^{d'})}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d'}\circ x'^{d'})}q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d'}\circ x'^{d'}|\boldsymbol{x}_0^{1:D})}$$

where $\circ$ represents the concatenation of a $D-1$ dimensional vector, $\boldsymbol{x}^{1:D\setminus d}$ with a scalar x^d, such that the resultant D dimensional vector has x^d at its d^{th} dimension. $\psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})$ is defined as the marginal of the forward noising process joint, $\int q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})r_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}^{1:D})d\boldsymbol{x}^{1:D}$.

Proof. We first re-write the general form of $\mathcal{L}_{\text{CT}}$ here

$$\mathcal{L}_{\text{CT}}(\theta) = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(x)r_t(\tilde{x}|x)}\left[\left\{\sum_{x'\neq x}\hat{R}_t^{\theta}(x,x')\right\} - \mathcal{Z}^t(x)\log\left(\hat{R}_t^{\theta}(\tilde{x},x)\right)\right] + C$$

where

$$\mathcal{Z}^t(x) = \textstyle\sum_{x'\neq x}R_t(x,x') \qquad\qquad r_t(\tilde{x}|x) = (1-\delta_{\tilde{x},x})R_t(x,\tilde{x})/\mathcal{Z}^t(x).$$

With a factorized forward process, $\hat{R}_t^{\theta}$ becomes

$$\hat{R}_t^{\theta\,1:D}(\boldsymbol{x}^{1:D},\tilde{\boldsymbol{x}}^{1:D}) = \sum_{d=1}^{D}\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},\tilde{x}^d)\delta_{\boldsymbol{x}^{1:D\setminus d},\tilde{\boldsymbol{x}}^{1:D\setminus d}}$$

where

$$\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},\tilde{x}^d) = R_t^d(\tilde{x}^d,x^d)\sum_{x_0^d}p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}^{1:D})\frac{q_{t|0}(\tilde{x}^d|x_0^d)}{q_{t|0}(x^d|x_0^d)}$$

Substituting this form for $\hat{R}_t^{\theta\,1:D}$ into the first term in $\mathcal{L}_{\text{CT}}$ we get

$$\sum_{\boldsymbol{x}'^{1:D}\neq\boldsymbol{x}^{1:D}}\sum_{d=1}^{D}\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},x'^d)\delta_{\boldsymbol{x}^{1:D\setminus d},\boldsymbol{x}'^{1:D\setminus d}}$$
$$= \sum_{d=1}^{D}\sum_{x'^d}\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},x'^d)\sum_{\boldsymbol{x}'^{1:D\setminus d}}\delta_{\boldsymbol{x}^{1:D\setminus d},\boldsymbol{x}'^{1:D\setminus d}}(1-\delta_{\boldsymbol{x}'^{1:D},\boldsymbol{x}^{1:D}})$$
$$= \sum_{d=1}^{D}\sum_{x'^d\neq x^d}\hat{R}_t^{\theta\,d}(\boldsymbol{x}^{1:D},x'^d)$$

Now we tackle the second term in $\mathcal{L}_{\text{CT}}$. We first re-arrange the distribution over which we take the expectation:

$$p_{\text{data}}(\boldsymbol{x}_0^{1:D})q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})r_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}^{1:D}) = p_{\text{data}}(\boldsymbol{x}_0^{1:D})\psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})\phi_t(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D},\boldsymbol{x}_0^{1:D})$$

We have,

$$\begin{aligned}\phi_t(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) &\propto q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D}) r_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}^{1:D}) \\ &= q_{t|0}(\boldsymbol{x}^{1:D}|\boldsymbol{x}_0^{1:D})(1 - \delta_{\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}}) \frac{\sum_{d=1}^D R_t^d(x^d, \tilde{x}^d) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}}}{\mathcal{Z}^t(\boldsymbol{x}^{1:D})} \\ &= \sum_{d=1}^D \frac{R_t^d(x^d, \tilde{x}^d)}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d)} q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d | \boldsymbol{x}_0^{1:D}) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} (1 - \delta_{\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}})\end{aligned}$$

To find the normalization constant, we can sum the proportional term over $\boldsymbol{x}^{1:D}$

$$\begin{aligned}&\sum_{\boldsymbol{x}^{1:D}} \sum_{d=1}^D \frac{R_t^d(x^d, \tilde{x}^d)}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d)} q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d | \boldsymbol{x}_0^{1:D}) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}} (1 - \delta_{\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}}) \\ &= \sum_{d=1}^D \sum_{x^d \neq \tilde{x}^d} \frac{R_t^d(x^d, \tilde{x}^d)}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d)} q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d | \boldsymbol{x}_0^{1:D})\end{aligned}$$

Therefore,

$$\phi_t(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) = (1 - \delta_{\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}}) \sum_{d=1}^D \phi_t(x^d|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) \delta_{\boldsymbol{x}^{1:D\setminus d}, \tilde{\boldsymbol{x}}^{1:D\setminus d}}$$

where

$$\phi_t(x^d|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) = \frac{R_t^d(x^d, \tilde{x}^d) q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d | \boldsymbol{x}_0^{1:D})}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d} \circ x^d) \sum_{d'=1}^D \sum_{x'^{d'} \neq \tilde{x}^{d'}} \frac{R_t^{d'}(x'^{d'}, \tilde{x}^{d'})}{\mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D\setminus d'} \circ x'^{d'})} q_{t|0}(\tilde{\boldsymbol{x}}^{1:D\setminus d'} \circ x'^{d'} | \boldsymbol{x}_0^{1:D})}$$

Now we write the second term as

$$\begin{aligned}&T\, \mathbb{E}_{t\sim\mathcal{U}(0,T) p_{\text{data}}(\boldsymbol{x}_0^{1:D}) \psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})} \left[- \sum_{\boldsymbol{x}^{1:D}} \phi_t(\boldsymbol{x}^{1:D}|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) \mathcal{Z}^t(\boldsymbol{x}^{1:D}) \log \hat{R}_t^{\theta\, 1:D}(\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}^{1:D}) \right] \\ &= -T\, \mathbb{E}_{t\sim\mathcal{U}(0,T) p_{\text{data}}(\boldsymbol{x}_0^{1:D}) \psi_t(\tilde{\boldsymbol{x}}^{1:D}|\boldsymbol{x}_0^{1:D})} \left[\sum_{d=1}^D \sum_{x^d \neq \tilde{x}^d} \phi_t(x^d|\tilde{\boldsymbol{x}}^{1:D}, \boldsymbol{x}_0^{1:D}) \mathcal{Z}^t(\tilde{\boldsymbol{x}}^{1:D/d} \circ x^d) \log \left(\hat{R}_t^{\theta\, d}(\tilde{\boldsymbol{x}}^{1:D}, x^d) \right) \right]\end{aligned}$$

□

C.4 One Forward Pass

To evaluate the $\mathcal{L}_{\text{CT}}$ objective, we naively need to perform two forward passes of the denoising network: $p_{0|t}^\theta(x_0|x)$ to calculate $\hat{R}_t^\theta(x, x')$ and $p_{0|t}^\theta(x_0|\tilde{x})$ to calculate $\hat{R}_t^\theta(\tilde{x}, x)$. This is wasteful because $\tilde{x}$ is created from x by applying a single forward transition which on multi-dimensional problems means $\tilde{x}$ differs from x in only a single dimension. To exploit the fact that $\tilde{x}$ and x are very similar, we approximate the sample $x \sim q_t(x)$ with the sample $\tilde{x} \sim \sum_x q_t(x) r_t(\tilde{x}|x)$. This gives the more efficient objective,

$$\mathcal{L}_{\text{eCT}}(\theta) = T\, \mathbb{E}_{t\sim\mathcal{U}(0,T) q_t(x) r_t(\tilde{x}|x)} \left[\left\{ \sum_{x' \neq \tilde{x}} \hat{R}_t^\theta(\tilde{x}, x') \right\} - \mathcal{Z}^t(x) \log \left(\hat{R}_t^\theta(\tilde{x}, x) \right) \right] + C$$

Table 3: Metrics on the monophonic music dataset comparing training with the efficient $\mathcal{L}_{\text{eCT}}$ objective vs the original $\mathcal{L}_{\text{CT}}$ objective. We compute these over the test set showing mean$\pm$std with respect to 5 samples for each test song.

Model	Hellinger Distance	Proportion of Outliers
τLDR-0 Uniform $\mathcal{L}_{\text{eCT}}$	0.3765 ± 0.0013	0.1106 ± 0.0010
τLDR-0 Uniform $\mathcal{L}_{\text{CT}}$	0.3797 ± 0.0009	0.1128 ± 0.0007

The approximation is valid because $q_t(x)$ and $\sum_x q_t(x) r_t(\tilde{x}|x)$ are very similar distributions, as we now show.

$$\begin{aligned}
\sum_x q_t(x) r_t(\tilde{x}|x) &= \sum_x q_t(x)(1-\delta_{x,\tilde{x}}) \frac{R_t(x,\tilde{x})}{\sum_{x' \neq x} R_t(x,x')} \\
&\propto -q_t(\tilde{x}) R_t(\tilde{x},\tilde{x}) + \sum_x q_t(x) R_t(x,\tilde{x}) \\
&= q_t(\tilde{x}) \sum_{x' \neq \tilde{x}} R_t(\tilde{x},x') + \partial_t q_t(\tilde{x}) \\
&\propto q_t(\tilde{x}) + \frac{1}{\sum_{x' \neq x} R_t(\tilde{x},x')} \partial_t q_t(\tilde{x}) \\
&= q_t(\tilde{x}) + \delta t \, \partial_t q_t(\tilde{x})
\end{aligned}$$

where on the third line we have used the Kolmogorov forward equation and defined $\delta_t = 1/\sum_{x' \neq x} R_t(\tilde{x},x')$. The distribution $\sum_x q_t(x) r_t(\tilde{x}|x)$ is therefore $q_{t+\delta t}(\tilde{x})$ approximated using a first-order Taylor expansion around $q_t(\tilde{x})$. We notice that δt is the average time to the next transition at time t. δt can be calculated for the practical settings we consider, its varies between $2\times10^{-6}T$ and $2\times10^{-8}T$ in the image modelling task and is $1\times10^{-3}T$ in the monophonic music task.

We perform an ablation experiment comparing between training with $\mathcal{L}_{\text{eCT}}$ and $\mathcal{L}_{\text{CT}}$ on the monophonic music dataset, the results are shown in Table 3. We find that we gain a small boost in performance when using the more efficient $\mathcal{L}_{\text{eCT}}$ objective alongside the improved efficiency. We hypothesize that this is because of a slight reduction in variance for the $\mathcal{L}_{\text{eCT}}$ objective due to increased negative correlation between the two terms in the objective when $\tilde{x}$ is shared between them.

D Direct Denoising Model Supervision

Following [8], we can introduce direct $p^\theta_{0|t}$ supervision into the optimization objective which has been found empirically to improve performance. We first contextualize the change by expressing $\mathcal{L}_{\text{CT}}$ with the dependence on $p^\theta_{0|t}$ made explicit.

$$\begin{aligned}
\mathcal{L}_{\text{CT}} = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T) q_t(x) r_t(\tilde{x}|x)} \Big[\Big\{ \sum_{x' \neq x} R_t(x',x) \sum_{x_0} \frac{q_{t|0}(x'|x_0)}{q_{t|0}(x|x_0)} p^\theta_{0|t}(x_0|x) \Big\} \\
- \mathcal{Z}^t(x) \log \Big(R_t(x,\tilde{x}) \sum_{x_0} \frac{q_{t|0}(x|x_0)}{q_{t|0}(\tilde{x}|x_0)} p^\theta_{0|t}(x_0|\tilde{x}) \Big) \Big] + C
\end{aligned}$$

The signal for $p^\theta_{0|t}(x_0|x)$ comes through a sum over x_0 weighted by the ratio $\frac{q_{t|0}(x|x_0)}{q_{t|0}(\tilde{x}|x_0)}$. We can also provide a direct denoising signal by predicting the clean datapoint x_0 from the corrupted version x and using the negative log-likelihood loss.

$$L_{ll}(\theta) = T\,\mathbb{E}_{t\sim\mathcal{U}(0,T) p_{\text{data}}(x_0) q_{t|0}(x|x_0)} \Big[-\log p^\theta_{0|t}(x_0|x) \Big]$$

Proposition 8. *The true denoising distribution, $q_{0|t}$, minimizes L_{ll}*

Proof.

$$
\begin{aligned}
&T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(x)}\left[\mathrm{KL}\left(q_{0|t}(x_0|x)\,||\,p^\theta_{0|t}(x_0|x)\right)\right] \\
&= T\,\mathbb{E}_{t\sim\mathcal{U}(0,T)p_{\text{data}}(x_0)q_{t|0}(x|x_0)}\left[-\log p^\theta_{0|t}(x_0|x)\right] + C
\end{aligned}
$$

where C is a constant independent of θ. Therefore, minimizing L_{ll} is equivalent to minimizing the KL divergence between $q_{0|t}$ and $p^\theta_{0|t}$, which is minimized when $p^\theta_{0|t} = q_{0|t}$. □

If we obtain the true denoising distribution, $p^\theta_{0|t} = q_{0|t}$, then we will have the true reverse rate, $\hat{R}_t$. [8] find that optimizing with an objective combining L_{ll} and $\mathcal{L}_{\text{DT}}$ performs best, which we can also do in continuous time

$$
\min_\theta \quad \mathcal{L}_{\text{CT}}(\theta) + \lambda L_{ll}(\theta)
$$

where λ is a hyperparameter. In [8], it was found that training with L_{ll} alone resulted in poorer performance than when the ELBO was included in the loss. We provide a theoretical hypothesis as to why this may be the case here. We show that minimizing L_{ll} is equivalent to minimizing an upper bound on the negative ELBO in discrete time and thus by training with L_{ll} we are simply minimizing a looser bound on the negative model log-likelihood than if we were to use the negative ELBO directly.

Proposition 9. *Minimizing the sum of negative log-likelihoods*

$$
\sum_{k=0}^{K-1} \mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)}\left[-\log p^\theta_{0|k+1}(x_0|x_{k+1})\right]
$$

is equivalent to minimizing an upper bound on the negative ELBO.

Proof.

$$
\begin{aligned}
\mathcal{L}_{\text{DT}}(\theta) = \mathbb{E}_{p_{\text{data}}(x_0)}\Big[&\mathrm{KL}(q_{K|0}(x_K|x_0)||p_{\text{ref}}(x_K)) - \mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p^\theta_{0|1}(x_0|x_1)\right] \\
&+ \textstyle\sum_{k=1}^{K-1} \mathbb{E}_{q_{k+1|0}(x_{k+1}|x_0)}\left[\mathrm{KL}(q_{k|k+1,0}(x_k|x_{k+1},x_0)||p^\theta_{k|k+1}(x_k|x_{k+1}))\right]\Big]
\end{aligned}
$$

Consider one term from the sum

$$
\begin{aligned}
L_k &= \mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)}\left[\mathrm{KL}(q_{k|k+1,0}(x_k|x_{k+1},x_0)||p^\theta_{k|k+1}(x_k|x_{k+1})\right] \\
&= \mathbb{E}_{q_{k+1}(x_{k+1})q_{k|k+1}(x_k|x_{k+1})}\left[-\log p^\theta_{k|k+1}(x_k|x_{k+1})\right] \\
&\quad + \mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)q_{k|k+1,0}(x_k|x_{k+1},x_0)}\left[\log q_{k|k+1,0}(x_k|x_{k+1},x_0)\right]
\end{aligned}
$$

Now,

$$
\begin{aligned}
&\mathbb{E}_{q_{k+1}(x_{k+1})q_{k|k+1}(x_k|x_{k+1})}\left[-\log p^\theta_{k|k+1}(x_k|x_{k+1})\right] \\
&= \mathbb{E}_{q_{k+1}(x_{k+1})q_{k|k+1}(x_k|x_{k+1})}\left[-\log \sum_{\tilde{x}_0} q(x_k|\tilde{x}_0,x_{k+1})p^\theta_{0|k+1}(\tilde{x}_0|x_{k+1})\right] \\
&= \mathbb{E}_{q_{k+1}(x_{k+1})q_{k|k+1}(x_k|x_{k+1})}\left[-\log \sum_{\tilde{x}_0} \frac{q_{0|k}(\tilde{x}_0|x_k)q_{k|k+1}(x_k|x_{k+1})}{q_{0|k+1}(\tilde{x}_0|x_{k+1})}p^\theta_{0|k+1}(\tilde{x}_0|x_{k+1})\right] \\
&\leq \mathbb{E}_{q_{k+1}(x_{k+1})q_{k|k+1}(x_k|x_{k+1})q_{0|k}(\tilde{x}_0|x_k)}\left[-\log \frac{q_{k|k+1}(x_k|x_{k+1})}{q_{0|k+1}(\tilde{x}_0|x_{k+1})}p^\theta_{0|k+1}(\tilde{x}_0|x_{k+1})\right] \\
&= \mathbb{E}_{p_{\text{data}}(x_0)q_{k+1|0}(x_{k+1}|x_0)}\left[-\log p^\theta_{0|k+1}(x_0|x_{k+1})\right] \\
&\quad + \mathbb{E}_{p_{\text{data}}(x_0)q_{k|0}(x_k|x_0)q_{k+1|k}(x_{k+1}|x_k)}\left[-\log \frac{q_{k|k+1}(x_k|x_{k+1})}{q_{0|k+1}(x_0|x_{k+1})}\right]
\end{aligned}
$$

Therefore,

$$\begin{aligned}\mathcal{L}_{\text{DT}} \leq \sum_{k=0}^{K-1} \left\{ \mathbb{E}_{p_{\text{data}}(x_0) q_{k+1|0}(x_{k+1}|x_0)} \left[-\log p^{\theta}_{0|k+1}(x_0|x_{k+1}) \right] \right\} \\ + \mathbb{E}_{p_{\text{data}}(x_0)} \left[\text{KL}(q_{K|0}(x_K|x_0)||p_K(x_K)) \right] \\ + \sum_{k=1}^{K-1} \left\{ \mathbb{E}_{p_{\text{data}}(x_0) q_{k+1|0}(x_{k+1}|x_0) q_{k|k+1,0}(x_k|x_{k+1},x_0)} \left[\log q_{k|k+1,0}(x_k|x_{k+1}, x_0) \right] \right. \\ \left. + \mathbb{E}_{p_{\text{data}}(x_0) q_{k|0}(x_k|x_0) q_{k+1|k}(x_{k+1}|x_k)} \left[-\log \frac{q_{k|k+1}(x_k|x_{k+1})}{q_{0|k+1}(x_0|x_{k+1})} \right] \right\}\end{aligned}$$

We can see that only the first term depends on θ. □

E Choice of Forward Process

We need to choose the structure of R_t such that we can analytically obtain $q_{t|0}$ marginals to enable efficient training.

Proposition 10. *If R_t and $R_{t'}$ commute for all t, t' then $q_{t|0}(x = j|x_0 = i) = \left(exp\left[\int_0^t R_s ds\right]\right)_{ij}$ where exp here is understood to be the matrix exponential function.*

Proof. Let $(P_t)_{ij} = q_{t|0}(x = j|x_0 = i)$. We show that $P_t = \exp\left(\int_0^t R_s ds\right)$ is a solution to the Kolmogorov forward equation, which in matrix form reads, $\partial_t P_t = P_t R_t$. Writing the matrix exponential in sum form

$$\begin{aligned}P_t &= \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t R_s ds \right)^k \\ &= \text{Id} + \int_0^t R_s ds + \frac{1}{2!} \left(\int_0^t R_s ds \right)^2 + \frac{1}{3!} \left(\int_0^t R_s ds \right)^3 + \dots\end{aligned}$$

Now, differentiating and using the fact that R_t, $R_{t'}$ commute.

$$\begin{aligned}\partial_t P_t &= R_t + \int_0^t R_s ds R_t + \frac{1}{2!} \left(\int_0^t R_s ds \right)^2 R_t + \dots \\ &= \left\{ \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t R_s ds \right)^k \right\} R_t \\ &= P_t R_t\end{aligned}$$

□

As stated in the main text, we achieve the commutative property by selecting $R_t = \beta(t) R_b$ where $\beta(t)$ is a time dependent scalar and R_b is a constant base matrix. We can utilize the eigendecomposition

of $R_b = Q\Lambda Q^{-1}$ to efficiently calculate P_t,

$$\begin{aligned}
P_t &= \exp\left(\int_0^t \beta(s) R_b ds\right) \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t \beta(s) R_b ds\right)^k \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} \left(Q\Lambda Q^{-1} \int_0^t \beta(s) ds\right)^k \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} Q \left(\Lambda \int_0^t \beta(s) ds\right)^k Q^{-1} \\
&= Q \left\{\sum_{k=0}^{\infty} \frac{1}{k!} \left(\Lambda \int_0^t \beta(s) ds\right)^k\right\} Q^{-1} \\
&= Q \exp\left[\Lambda \int_0^t \beta(s) ds\right] Q^{-1}
\end{aligned}$$

Since Λ is a diagonal matrix, the matrix exponential coincides with the element wise exponential making the final expression tractable to compute. We choose $\beta(t) = ab^t \log b$ because this makes the integral which dictates the variance of $q_{t|0}$ have a simple form $\int_0^t \beta(s) ds = ab^t - a$.

For categorical problems, we found a uniform rate matrix works well, $R_t = \beta \mathbb{1}\mathbb{1}^T - \beta S \mathrm{Id}$. This is directly analogous to the discrete time uniform transition matrix: $P = \alpha \mathbb{1}\mathbb{1}^T + (1 - S\alpha)\mathrm{Id}$ with α depending on the time discretization used. Indeed, if one calculates the corresponding discrete transition matrix for the uniform R_t rate through the matrix exponential, the uniform transition matrix is obtained. Another categorical corruption process is the absorbing state process. In discrete time, the transition matrix is given by $P = \alpha \mathbb{1}\mathbf{e}_*^T + (1 - \alpha)\mathrm{Id}$ where $\mathbf{e}_*$ is the one-hot encoding of the absorbing state. The corresponding absorbing state continuous time process has transition rate matrix: $R_t = \beta \mathbf{1}\mathbf{e}_*^T - \beta \mathrm{Id}$. The correspondence for more complex transition matrices e.g. the Discretized Gaussian matrix in [8] is much harder to find analytically especially if the time inhomogeneous case is considered. For datasets with an ordinal structure, we construct a new rate matrix that maintains a bias towards nearby states using a similar approach as that taken by [8] to construct the Discretized Gaussian matrix.

We construct this matrix by first picking a desired stationary distribution, p_{ref}, and then filling in matrix entries such that we encourage transitions to nearby states whilst keeping p_{ref} as our stationary distribution. Specifically, we let p_{ref} be a discretized Gaussian over the state space, i.e.

$$p_{\text{ref}}(x) \propto \exp\left[-\frac{(x - \mu_0)^2}{2\sigma_0^2}\right]$$

To find a condition on the rate such that this is the case, recall the Kolmogorov differential equation for the marginals

$$\partial_t q_t(x) = \sum_{\tilde{x}} q_t(\tilde{x}) R_b(\tilde{x}, x)$$

Now, consider a rate that is in detailed balance with p_{ref}

$$p_{\text{ref}}(\tilde{x}) R_b(\tilde{x}, x) = p_{\text{ref}}(x) R_b(x, \tilde{x})$$

Substituting this rate into the Kolmogorov equation, we see that p_{ref} is the stationary distribution

$$\begin{aligned}
\partial_t p_{\text{ref}}(x) &= \sum_{\tilde{x}} p_{\text{ref}}(\tilde{x}) R_b(\tilde{x}, x) \\
&= \sum_{\tilde{x}} p_{\text{ref}}(x) R_b(x, \tilde{x}) \\
&= p_{\text{ref}}(x) \sum_{\tilde{x}} R_b(x, \tilde{x}) \\
&= 0
\end{aligned}$$

where the last line follows from the fact that the row sum of a rate matrix is zero. Note that any $R_t = \beta(t)R_b$ will also have this stationary distribution as the multiplication by $\beta(t)$ can be seen as just a scaling of the time axis. From the detailed balance equation, we gain a condition on R_b such that our desired p_{ref} is the stationary distribution

$$\frac{R_b(\tilde{x}, x)}{R_b(x, \tilde{x})} = \frac{p_{\text{ref}}(x)}{p_{\text{ref}}(\tilde{x})} = \exp\left[\frac{(\tilde{x} - \mu_0)^2}{2\sigma_0^2} - \frac{(x - \mu_0)^2}{2\sigma_0^2}\right]$$

This gives constraints on diagonal elements within R_b but does not fully define the entire matrix. To do this, we first make the assumption that μ is selected to be at the center of the state space. Then we set off diagonal terms to the right of the diagonal in the top half of the rate matrix and off diagonal terms to the left of the diagonal in the bottom half to be 1. Finally, progressing in from the top and bottom of the rate matrix we make definitions of rate matrix values that have not already been defined by the detailed balance condition. For clarity, we provide a pictorial representation of this scheme for an 8×8 rate matrix below

$$\begin{bmatrix} \cdot & 1 & \square & \square & \square & \square & \square & \square \\ \triangle & \cdot & 1 & \square & \square & \square & \square & \triangle \\ \triangle & \triangle & \cdot & 1 & \square & \square & \triangle & \triangle \\ \triangle & \triangle & \triangle & \cdot & 1 & \triangle & \triangle & \triangle \\ \triangle & \triangle & \triangle & \triangle & \cdot & \triangle & \triangle & \triangle \\ \triangle & \triangle & \triangle & \square & 1 & \cdot & \triangle & \triangle \\ \triangle & \triangle & \square & \square & \square & 1 & \cdot & \triangle \\ \triangle & \square & \square & \square & \square & \square & 1 & \cdot \end{bmatrix}$$

where $\square$ represents a value we will define, $\triangle$ represents a value that is defined relative to another entry through the detailed balance condition and $\cdot$ is a diagonal entry that is equal to the negative off diagonal row sum. We could define $\square$ values to be 0 to gain a sparse rate matrix, however, we found in early experiments that allowing transitions to further away states greatly reduces the mixing time and gives better performance. We define $\square$ in each row similarly, by setting it equal to $\exp[-i^2/\sigma_r^2]$ where i is the distance away from the '1' value in that row and σ_r is a hyperparameter defining the length scale in state space of a typical transition. This biases our forward process to make transitions between nearby states, at a length scale of σ_r.

F CTMC Simulation

F.1 Exact CTMC and Tau-Leaping

In this section, we first describe exact CTMC simulation before giving an algorithmic description of tau-leaping.

When a CTMC has a time-homogeneous rate matrix, we can use Gillespie's Algorithm [21, 22, 23] to exactly simulate it. This algorithm is based on the jump chain/holding time definition of the CTMC. It repeats the following two steps:

- Draw a holding time from an exponential distribution with mean $-1/R(x, x)$ and wait in the current state x for that amount of time.
- Sample the next state from $r(\tilde{x}|x) = (1 - \delta_{x,\tilde{x}})\frac{R(x,\tilde{x})}{\sum_{x' \neq x} R(x,x')}$

This Algorithm can be adjusted for the case when we have a time-inhomogeneous rate matrix using the modified next reaction method [33]. However, both algorithms still step through each transition in the CTMC individually and are thus unsuitable in our case because only one dimension would change for each simulation step making it very computationally expensive to produce a sample. Instead we use tau-leaping that allows multiple dimensions to change in a single simulation step. We detail this method in Algorithm 1.

F.2 Predictor-Corrector Discussion

In this section we compare predictor-corrector sampling schemes as applied to continuous state spaces and discrete state spaces.

Algorithm 1: Generative Reverse Process Simulation with Tau-Leaping

$t \leftarrow T$
$\boldsymbol{x}_t^{1:D} \sim p_{\text{ref}}(\boldsymbol{x}_T^{1:D})$
while $t > 0$ **do**
  Compute $p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}_t^{1:D}), d = 1, \ldots, D$ with one forward pass of the denoising network
  for $d = 1, \ldots, D$ **do**
    for $s = 1, \ldots, S \backslash x_t^d$ **do**
      $\hat{R}_t^{\theta\, d}(\boldsymbol{x}_t^{1:D}, s) \leftarrow R_t^d(s, x_t^d) \sum_{x_0^d} p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}_t^{1:D}) \frac{q_{t|0}(s|x_0^d)}{q_{t|0}(x_t^d|x_0^d)}$
      $P_{ds} \leftarrow \text{Poisson}\Big(\tau \hat{R}_t^{\theta\, d}(\boldsymbol{x}_t^{1:D}, s)\Big)$
    end
  end
  for $d = 1, \ldots, D$ **do**
    if *data is categorical AND* $\sum_{s=1}^{S} P_{ds} > 1$ **then**
      $x_{t-\tau}^d \leftarrow x_t^d$ `// reject change`
    else
      $x_{t-\tau}^d \leftarrow x_t^d + \sum_{s=1}^{S} P_{ds} \times (s - x_t^d)$
    end
  end
  $\boldsymbol{x}_{t-\tau}^{1:D} \leftarrow \text{Clamp}(\boldsymbol{x}_{t-\tau}^{1:D}, \text{min} = 1, \text{max} = S)$
  $t \leftarrow t - \tau$
end

The predictor-corrector scheme in continuous state spaces was introduced in [4]. It consists of alternating between a predictor step and a corrector step:

$$\text{Predictor} \quad \boldsymbol{x}_i \leftarrow \boldsymbol{x}_{i+1} + \gamma_i s_\theta(\boldsymbol{x}_{i+1}, i+1) + \sqrt{\gamma_i} z, \quad z \sim \mathcal{N}(0, I)$$
$$\text{Corrector} \quad \boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon_i s_\theta(\boldsymbol{x}_i, i) + \sqrt{2\epsilon_i} z, \quad z \sim \mathcal{N}(0, I)$$

where $\boldsymbol{x}_i$ is the state at sampling step i, s_θ is the learned score model approximating $\nabla_{\boldsymbol{x}} \log q_t(\boldsymbol{x})$ and γ_i, ϵ_i are the step sizes for the predictor and corrector respectively. We see that both take similar forms, except the corrector adds in a factor $\sqrt{2}$ more Gaussian noise during the update step.

In discrete state spaces, rather than sampling using gradient guided stochastic steps as in the continuous state space case, we sample by simulating CTMCs with defined rates. When we take a predictor step, we simulate using $\hat{R}_t^\theta$ and when we take a corrector step we simulate using $R_t^{c\,\theta} = \hat{R}_t^\theta + R_t$. If we simulate the CTMC exactly, we have seen in the previous section that this amounts to sampling next states from the categorical distribution defined by normalizing the row of the rate matrix corresponding to the current state. Therefore, corrector sampling can be seen as sampling from a slightly noisier categorical distribution defined through $R_t^{c\,\theta}$ as compared to the predictor categorical distribution defined through $\hat{R}_t^\theta$. This is analogous to the increased Gaussian noise applied during a corrector step in continuous state spaces.

Adding corrector steps brings the marginal of the samples closer to $q_t(\boldsymbol{x})$ and continued application of the corrector will further explore the domain of $q_t(\boldsymbol{x})$. In previous work on continuous state predictor-corrector methods, the number of corrector steps has been small (e.g. 1 or 2 corrector steps per predictor step) or indeed the corrector steps have been removed altogether. In this work we have found that using up to 10 corrector steps per predictor steps can be beneficial during certain regions of the reverse generative process. Additionally, in continuous state spaces, it has been observed that too many corrector steps can result in unwanted noise in the generated data [34].

We hypothesize that corrector steps are better utilized in discrete state spaces to explore the domain of $q_t(\boldsymbol{x})$ than in continuous state spaces. This is because, the corrector update is defined largely through the reverse rate itself, $\hat{R}_t^\theta$, just with the categorical probabilities being annealed slightly more towards uniform through the addition of the forward rate R_t. This may be a more effective update than simply adding extra Gaussian noise in the continuous state space case. Furthermore, the denoising model in continuous state spaces can be seen as outputting a point estimate of $\boldsymbol{x}_0$ of dimension D. However,

Figure 7: Progression of $\boldsymbol{x}_t$ for $t = 0.4$ by repeated application of corrector steps. In each pair of rows, the top row is $\boldsymbol{x}_t$ whilst the bottom row is the $\boldsymbol{x}_0$ prediction made by $p^\theta_{0|t}(\boldsymbol{x}_0|\boldsymbol{x}_t)$ (argmax of the categorical probabilities in each dimension). Each column represents an additional 100 corrector steps.

in discrete state spaces, the denoising model outputs a categorical distribution over every dimension (output dimension $D \times S$) allowing it to express some uncertainty information in the $\boldsymbol{x}_0$ prediction, albeit with conditional independence between the dimensions. Adding corrector steps in discrete state spaces would then allow information to mix between dimensions for the current time step, exploring modes of $q_t(\boldsymbol{x})$.

We explore this idea on the image modelling task in Figure 7. We run the reverse generative process until time $t = 0.4$ at which point we hold the time constant and apply 1000 corrector steps. We see that the resulting progression of $\boldsymbol{x}_t$ states explores potential local modes of $q_t(\boldsymbol{x})$ in the local region of image space.

G Implicit Dimensional Assumptions Made in Discrete Time

In discrete time, the parametric reverse kernel, $p^\theta_{k|k+1}$, is commonly defined through a denoising model $p^\theta_{0|k+1}$. Here, we examine this definition in the multi-dimensional case where the forward process factorizes, as in Appendix C.3 and previous discrete time work [8]. We begin by writing the true full dimensional reverse kernel, $q_{k|k+1}$, in terms of the true denoising distribution, $q_{0|k+1}$.

$$
\begin{aligned}
q_{k|k+1}(\boldsymbol{x}_k^{1:D}|\boldsymbol{x}_{k+1}^{1:D}) &= \prod_{d=1}^{D} q_{k|k+1}(x_k^d|\boldsymbol{x}_k^{1:d-1}, \boldsymbol{x}_{k+1}^{1:D}) \\
&= \prod_{d=1}^{D} \sum_{x_0^d} q_{k,0|k+1}(x_k^d, x_0^d|\boldsymbol{x}_k^{1:d-1}, \boldsymbol{x}_{k+1}^{1:D}) \\
&= \prod_{d=1}^{D} \sum_{x_0^d} q_{0|k+1}(x_0^d|\boldsymbol{x}_k^{1:d-1}, \boldsymbol{x}_{k+1}^{1:D}) q_{k|0,k+1}(x_k^d|x_0^d, x_{k+1}^d)
\end{aligned}
$$

where on the final line we have used the fact that the forward process is independent across dimensions. To create our approximate reverse kernel, $p^\theta_{k|k+1}$, we approximate $q_{0|k+1}(x_0^d|\boldsymbol{x}_k^{1:d-1}, \boldsymbol{x}_{k+1}^{1:D})$ with

$p^\theta_{0|k+1}(x_0^d|\boldsymbol{x}_{k+1}^{1:D})$,

$$p^\theta_{k|k+1}(\boldsymbol{x}_k^{1:D}|\boldsymbol{x}_{k+1}^{1:D}) = \prod_{d=1}^{D} \sum_{x_0^d} p^\theta_{0|k+1}(x_0^d|\boldsymbol{x}_{k+1}^{1:D}) q_{k|0,k+1}(x_k^d|x_0^d, x_{k+1}^d)$$

We throw away the extra $\boldsymbol{x}_k^{1:d-1}$ conditioning because we use a non-autoregressive model that takes in $\boldsymbol{x}_{k+1}^{1:D}$ and in a single forward pass gives conditionally independent probabilities over x_0^d, $d = 1, \dots, D$. For finite K, this approximation can never match the true kernel because we are not conditioning on all relevant information. Of course, as K gets larger, this approximation becomes more accurate. Since we operate in the continuous regime, we do not have to make this approximation because the conditionally independent denoising model, $q_{0|t}(x_0^d|\boldsymbol{x}^{1:D})$, appears directly in our reverse rate, $\hat{R}_t^{1:D}$, when we factorize the forward process (see Proposition 3).

H Experimental Details

In this section, we provide additional details for the experiments we performed applying our method to practical problems. The code for our models is available at `https://github.com/andrew-cr/tauLDR`. Before describing the specifics for each experiment, we first explain the implementation details common to all.

When we evaluate the objective $\mathcal{L}_{\text{CT}}$ on each minibatch of training datapoints, we must sample a time for each from $t \sim \mathcal{U}(0, T)$ which represents the point in the forward process which we will noise to. Training instabilities can be found if t is sampled very close to 0 because the reverse rate, $\hat{R}_t$, becomes ill-conditioned in this region. This phenomenon is also observed in continuous state space models because the score, $\nabla_x \log q_t(x)$, becomes ill-conditioned close to $t = 0$. The reverse rate and score become ill conditioned close to the start of the forward process because the marginal probability, $q_t(x)$, will be highly peaked around the data manifold and $\log q_t(x)$ will explode in regions that are not close to the data. To avoid these issues, a common trick is to set a minimum time such that $t \sim \mathcal{U}(\epsilon, T)$. ϵ is set such that the level of noising at $t = \epsilon$ is very small and reverse sampling to this point will produce samples very close to p_{data}. In our experiments, we set $\epsilon = 0.01T$.

During reverse sampling, we use tau-leaping to simulate the reverse process from $t = T$ until $t = \epsilon$ because the reverse rate is not trained for $t < \epsilon$. This produces a sample close to p_{data}. We found improved performance in metrics such as FID if we then complete a final step to remove the small amount of noise that may still be present in the sample. Specifically, we pass the sample through the denoising model $p^\theta_{0|t}(x_0|x_t)$ with $t = \epsilon$ to obtain an output of shape $D \times S$ where D is the dimensionality of the problem. This is a probability distribution over the states for each of the dimensions. We set the value of each dimension to the state with the highest probability. This then produces a sample which has all of the noise removed.

The specific value of T within our model is arbitrary because the forward process can be scaled in the time axis to provide the same noising process for any T. Therefore, we simply set $T = 1$.

H.1 Demonstrative Example

Our 2d dataset is created by sampling 1M 2d points from a 32×32 state space with probability proportional to the pixel values of a 32×32 grayscale image of a τ character.

For our forward process, we use a Gaussian rate (see Appendix E) with stationary distribution standard deviation $\sigma_0 = 8$ and rate length scale $\sigma_r = 1$. We use a rate schedule of $\beta(t) = 5 \times 5^t \log(5)$.

To represent $p^\theta_{0|t}$ we use a residual MLP. The architecture consists of an input linear layer to lift the input dimension of 2 to the internal network dimension of 16. Then, there are 2 residual blocks each consisting of: a single hidden layer MLP of hidden dimension 32, a residual connection to the

input of the MLP, a layer norm, and finally a FiLM layer [35] modulated by the time embedding. At the output, there is a single linear layer with output size of $2 \times 32 = 64$ representing state probabilities in each of the 2 dimensions. The time is embedded using the Transformer sinusoidal position embedding [36] creating an embedding of size 32. Then, the embedding is further processed by a single hidden layer MLP with hidden layer size 32 and output size 128. To create the FiLM parameters in each residual block, the time embedding is passed through a linear layer with output of size 32 to provide a multiplicative and additive modulation to the state dimension of 16. We minimize the $\mathcal{L}_{\text{CT}}$ objective using Adam with a learning rate of 0.0001 and batch size of 32 for 1M steps.

For the exact simulation we use the next reaction method with modifications for time dependent transition rates [33]. This method steps through each transition in the exact simulation path individually by calculating the time to the next occurrence of each transition type and applying the transition that occurs soonest. Exact algorithmic details can be found in [33]. To calculate the time to the next occurrence for a transition, we need to integrate the reverse rate matrix (eq (13) in [33]). We do this with euler integration with a step size of 0.001.

H.2 Image Modeling

We train on the CIFAR10 training dataset that contains 50000 images of dimension $3 \times 32 \times 32$. We evaluate the test ELBO on the CIFAR10 test dataset which consists of 10000 images. For the forward noising process, we use the the Gaussian rate (see Appendix E) with stationary distribution standard deviation of $\sigma_0 = 512$ and rate length scale $\sigma_r = 6$. This effectively defines a uniform stationary distribution since the state space is of size 256. We found this performs better than a more concentrated Gaussian. Our β schedule is $\beta(t) = 3 \times 100^t \log 100$. This was selected in accordance with σ_r such that the overall shape of progression of the $q_{t|0}$ variances approximately matches that of the schedule proposed in [3].

Our $p^\theta_{0|t}$ model is parameterized with the standard U-net [37] architecture introduced in [3]. The network follows the PixelCNN++ backbone [38] with group normalization layers. There are four feature map resolutions (32×32 to 4×4) in the downsampling/upsampling stacks. At each resolution there are two convolutional residual blocks. There is a self-attention block between the residual blocks at the 16×16 resolution level [39]. The time is input into the network by first embedding with the Transformer sinusoidal position embedding [36]. This time embedding is passed into each residual block by passing it through a SiLU activation [40] and then a linear layer before adding it onto the hidden state within the residual block between the two convolution operations.

The original architecture of [3] has an output of dimension $3 \times 32 \times 32$ as it makes a point prediction of x_0 given x_t. In order for the model to output probabilities over x_0 (i.e. an output dimension of $3 \times 32 \times 32 \times 256$) we make the adjustments suggested in [8]. Specifically, we use their truncated logistic distribution parameterization where the model outputs the mean and log scale of a logistic distribution i.e. an output dimension of $3 \times 32 \times 32 \times 2$. The probability for a state is then the integral of this continuous distribution between this state and the next when mapped onto the real line. To impart a residual inductive bias on the output, the mean of the logistic distribution is taken to be $\tanh(x_t + \mu')$ where x_t is the normalized input into the model and μ' is mean outputted from the network. The normalization operation takes the input in the range $0, \dots, 255$ and maps it to $[-1, 1]$. In total, our network has approximately 35.7 million parameters.

We optimize with the auxiliary objective described in Appendix D with $\lambda = 0.001$. Within the auxiliary objective, we use the one-forward pass version of the continuous time ELBO, $\mathcal{L}_{\text{eCT}}$. We optimize with Adam for 2M steps with a learning rate of 0.0002 and batch size of 128. We use the standard set of training tricks to improve optimization [3, 4]. Throughout training we maintain an exponential moving average of the parameters with decay factor 0.9999. These average parameters are used during testing. At the start of optimization we use a linear learning rate warm-up for the first 5000 steps. We clip the gradient norm at a norm value of 1.0. We set the dropout rate for the network at 0.1. The skip connections for each residual block are rescaled by a factor of $\frac{1}{\sqrt{2}}$. The input images have random horizontal flips applied to them during training.

For sampling in Table 1 we set $\tau = 0.001$ for τLDR-0 and set $\tau = 0.002$ for τLDR-10. The 10 corrector steps per predictor steps for τLDR-10 are introduced after $t < 0.1T$. We found that introducing the corrector steps near the end of the reverse sampling process had the best improvement in sample quality for the smallest increase in computational cost. When performing tau-leaping with the corrector rate, R_t^c, we have control over what τ we use since we are sampling a different CTMC (with q_t as its stationary distribution) to the original reverse CTMC. We found that setting the corrector rate τ to be 1.5 times the original τ for the reverse CTMC achieves the best performance in this example.

We train using 4 V100 GPUs on an academic research cluster. To calculate Inception and FID values, we use pytorch-fid [41] and a further development [1]. We verified this library produced comparable values to previous work by calculating the Inception and FID scores for the published images from the DDPM [3] method.

We show a large array of unconditional samples from the τLDR-10 model in Figure 8. We now also present statistics from the reverse sampling process with standard tau-leaping with $\tau = 0.001$. Figure 9 shows the proportion of dimensions that transition during a single step of tau-leaping. We see that during the initial stages, every dimensions changes during every tau-leaping step, but nearer the end of the process, more dimensions will have settled in their final positions and the proportion is less. In Figure 10 we show the proportion of dimensions that are clipped due to proposing an out of bounds jump. Overall, the proportion is small. It is largest at the start of the process when we have initially sampled from the approximately uniform p_{ref} and there will be dimensions close to the boundary. As pixel values settle to their final values, the proportion reduces. Figure 11 shows the progression of a selection of dimensions during the reverse sampling process. A similar picture emerges where dimensions eventually settle in a region of the state space. We also note that larger jumps are made in a single tau-leaping step nearer the start of the reverse process and smaller jumps are made nearer the end.

H.3 Monophonic Music

We generate our training dataset from the Lakh pianoroll dataset [29, 30] (license CC By 4.0). This dataset consists of 174,154 multitrack pianorolls. We go through all songs and all tracks within each song and select sequences that match the following criteria: they are monophonic (only one note played at a time), there is not a period longer than one bar in which no note is played, there is more than one type of note played in the sequence and finally there is no one note played for more than 50 time steps out of the total 256 time steps. This removes the uninteresting and trivial sequences present within the dataset. We then remove any duplicates in the result. This leaves us with 6000 training examples and 950 testing examples. Each song consists of 256 time steps (16 per bar) and each time step takes one from 129 values i.e. we have $D = 256$ and $S = 129$. This state value represents either a note from 128 options or a rest. We scramble the ordering of this state space when mapping to the integers from 0 to 128. When we input into the denoising network, we input as one-hot 129 dimensional vectors.

For the forward noising process, we use a uniform rate matrix, $R_b = \mathbb{1}\mathbb{1}^T - S\text{Id}$ and set $\beta(t) = 0.03$. We found a constant in time $\beta(t)$ was sufficient for this dataset. In our comparison, we used a birth/death rate matrix defined as

$$\begin{bmatrix} -\lambda & \lambda & 0 & 0 & \dots & 0 & 0 \\ \lambda & -2\lambda & \lambda & 0 & \dots & 0 & 0 \\ 0 & \lambda & -2\lambda & \lambda & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \lambda & -\lambda \end{bmatrix}$$

this is the rate matrix for a birth/death process. We set $\lambda = 1$ and $\beta(t) = \frac{1}{2} \times 10000^t \log 10000$. These hyperparameters were selected such that the forward process has a steady rate of noising whilst still having q_T very close to p_{ref}. We chose to compare these types of rate matrix because the birth/death rate is inappropriate for this categorical data as adjacent states have no meaning

[1] `https://github.com/w86763777/pytorch-gan-metrics`

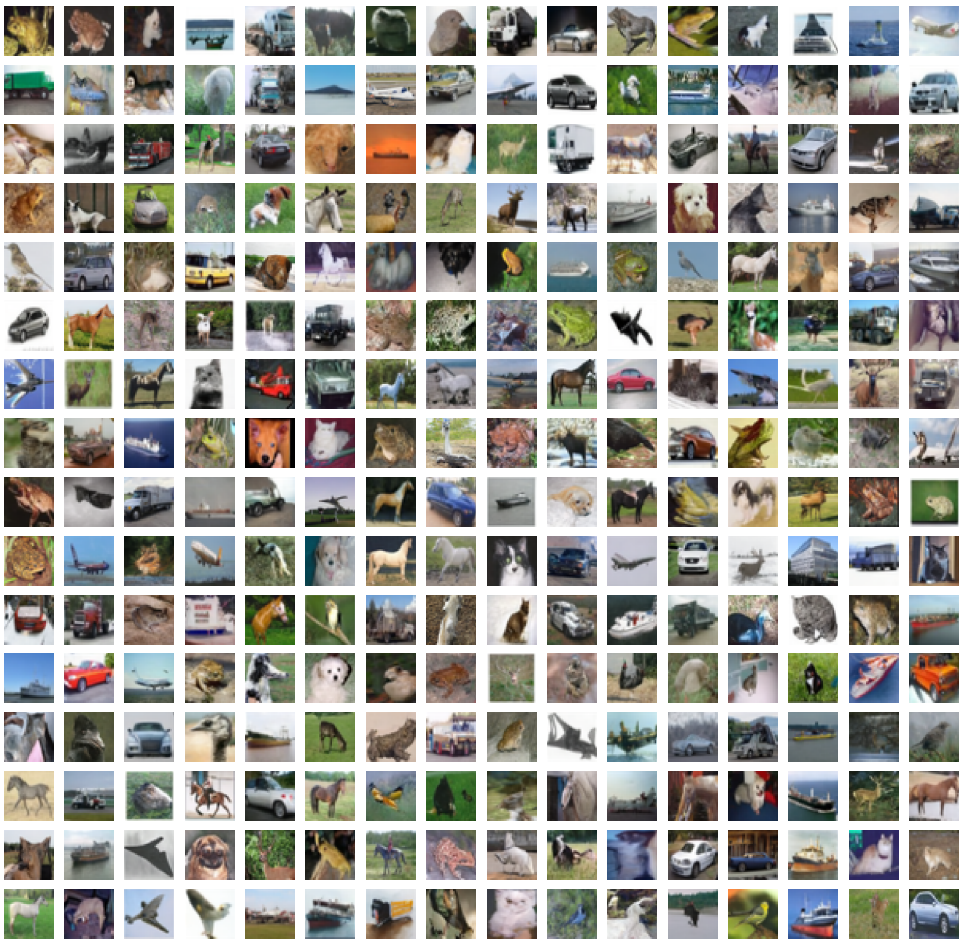


Figure 8: Unconditional CIFAR10 samples from our τLDR-10 model.

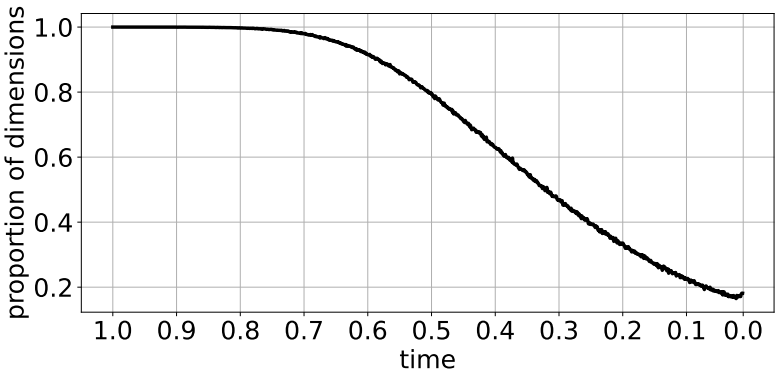


Figure 9: Proportion of dimensions that transition during a single step of tau-leaping during the reverse sampling process.

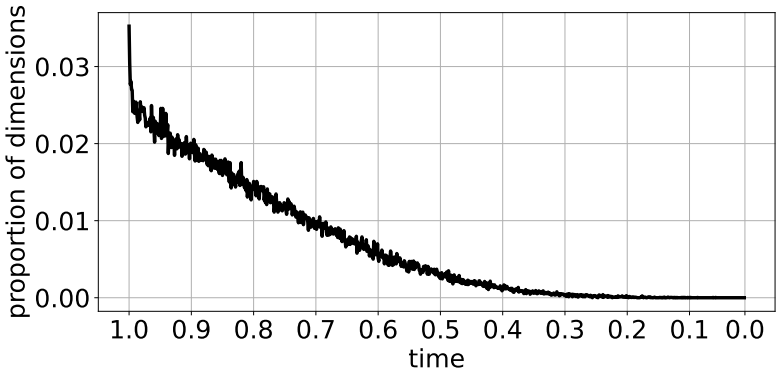


Figure 10: Proportion of dimensions that are clipped during a tau-leaping step due to proposing an out of bounds jump during the reverse sampling process.

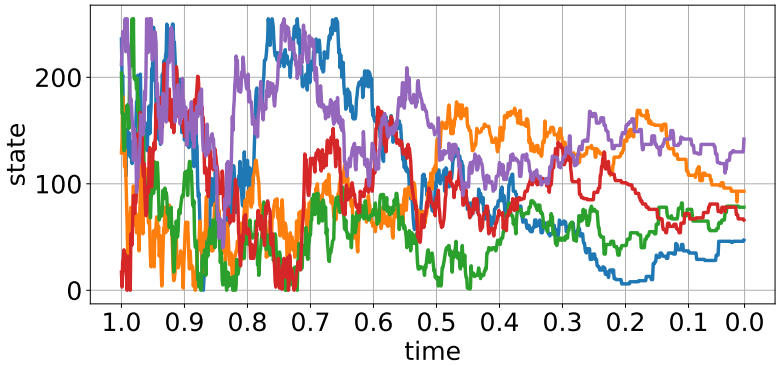


Figure 11: The progression of a selection of dimensions during the reverse sampling process.

since the mapping to the integers was arbitrary. The uniform rate is suitable for this categorical data because, during a time interval, it has a uniform probability to transition to any other state. The D3PM baseline was implemented also with a time homogeneous uniform forward kernel set such that the rate of noising is matched in the discrete and continuous time cases.

We define our conditional denoising network, $p^{\theta}_{0|t}(x_0|x, y)$ using a transformer architecture inspired by [31]. It takes an input of shape (B, D, S) where B is the batch size, D is the dimensionality (256) and S is the state size (129). This final dimension contains the one-hot vectors. The conditioning on the initial bars is achieved by concatenating the conditioning information y with the noisy input x. At the start of the network, there is an input embedding linear layer with output of size 128 which is our model dimension for the transformer. Then a transformer positional embedding is added to the hidden state. Next a stack of 6 transformer encoder layers are applied which consist of a self attention block and a one hidden layer MLP. The self attention block uses 8 heads and the MLP has a hidden layer size of 2048. At the output of each internal block, we apply dropout with rate 0.1. Finally, there is a stack of 2 residual MLP layers. Each consists of a one hidden layer MLP with a hidden dimension of 2048. There is a residual connection between the input and output of the MLP. A layer norm is applied to the output of the block. To create the output of the network, there is an output linear layer with an output shape of (B, D, S) where now the S dimension has logit probabilities. To instill a residual bias into the network, we add the one-hot input to the output logits. All activations are ReLU. The time is input into the network through FiLM layers [35]. First, the time is embedded using the sinusoidal transformer position embedding as in the U-net architecture used for image modeling to create an embedding size of 128. This is then passed into a single hidden layer MLP with hidden size

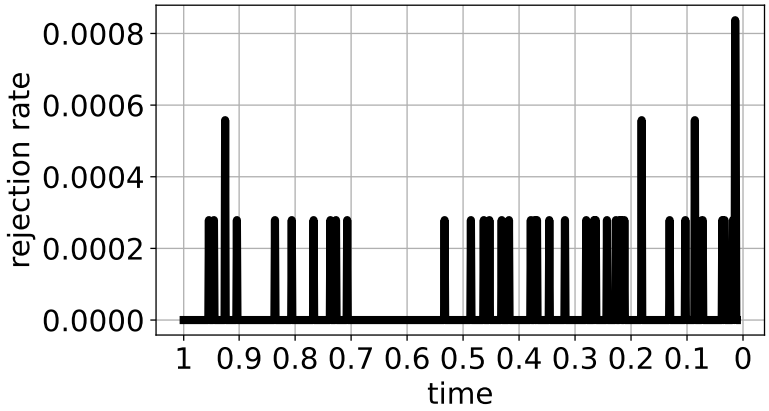


Figure 12: Proportion of jumps rejected during reverse sampling. The rejection rate is calculated as the proportion of dimensions in a tau leaping step that have their jump rejected. The results are averaged over a batch of 16.

2048 and output size 512. Within each encoder and residual MLP block, there is a FiLM linear layer which takes in the 512 time embedding and outputs two FiLM parameters each of size 128. These are the scale and offset applied to the hidden state. In the encoder blocks, this FiLM transform is applied after the self attention block and again after the fully connected block. In the residual MLP blocks, it is applied after the layer norm operation. Our network has approximately 7 million parameters in total.

We optimize using Adam for 1M steps with a batch size of 64 and learning rate of 0.0002. We use the conditional $\bar{\mathcal{L}}_{\text{CT}}$ objective with additional direct $p^{\theta}_{0|t}$ supervision as described in Appendix D with weight $\lambda = 0.001$. We also make the same one forward pass approximation as explained in Appendix C.4. We use the standard set of training tricks to improve optimization [3, 4]. Throughout training we maintain an exponential moving average of the parameters with decay factor 0.9999. These average parameters are used during testing. At the start of optimization we use a linear learning rate warm-up for the first 5000 steps. We clip the gradient norm at a norm value of 1.0. We train on a single V100 GPU on an academic cluster.

For sampling with τLDR-0 we use $\tau = 0.001$ and for sampling with τLDR-2 we include 2 corrector steps per predictor step after $t < 0.9T$. The corrector rate is simulated with $\tau = 0.0001$ which we found to perform best. We reject any dimension in which 2 or more jumps are proposed as this is categorical data. We plot the rejection rate in Figure 12. Most of the time, the rejection rate is zero and there are few steps for which it increases slightly. We show a large batch of samples from the first 10 songs in the test dataset in Figure 13. We see that there is variation between the sampled completions and they consistently follow the style of the conditioning first two bars of the song. Audio samples from the model are available at `https://github.com/andrew-cr/tauLDR`. Finally, we examine the progression of a random selection of dimensions during reverse sampling for the uniform and birth/death rate matrix cases. Figure 14 shows the progression for the uniform case, we see that large jumps through the state space are made throughout the reverse process. Figure 15 shows the progression for the birth/death case. At the start of reverse sampling, no dimensions move as the rejection rate is high in this case because the rate matrix is not suitable for categorical data. Nearer the end, small jumps are made between adjacent states but since large jumps between any category do not occur for this rate matrix, the performance will overall be worse.

I Ethical Considerations

Our work increases our theoretical understanding of denoising generative models and also improves generation capabilities within some discrete datasets. Deep generative models are generic methods

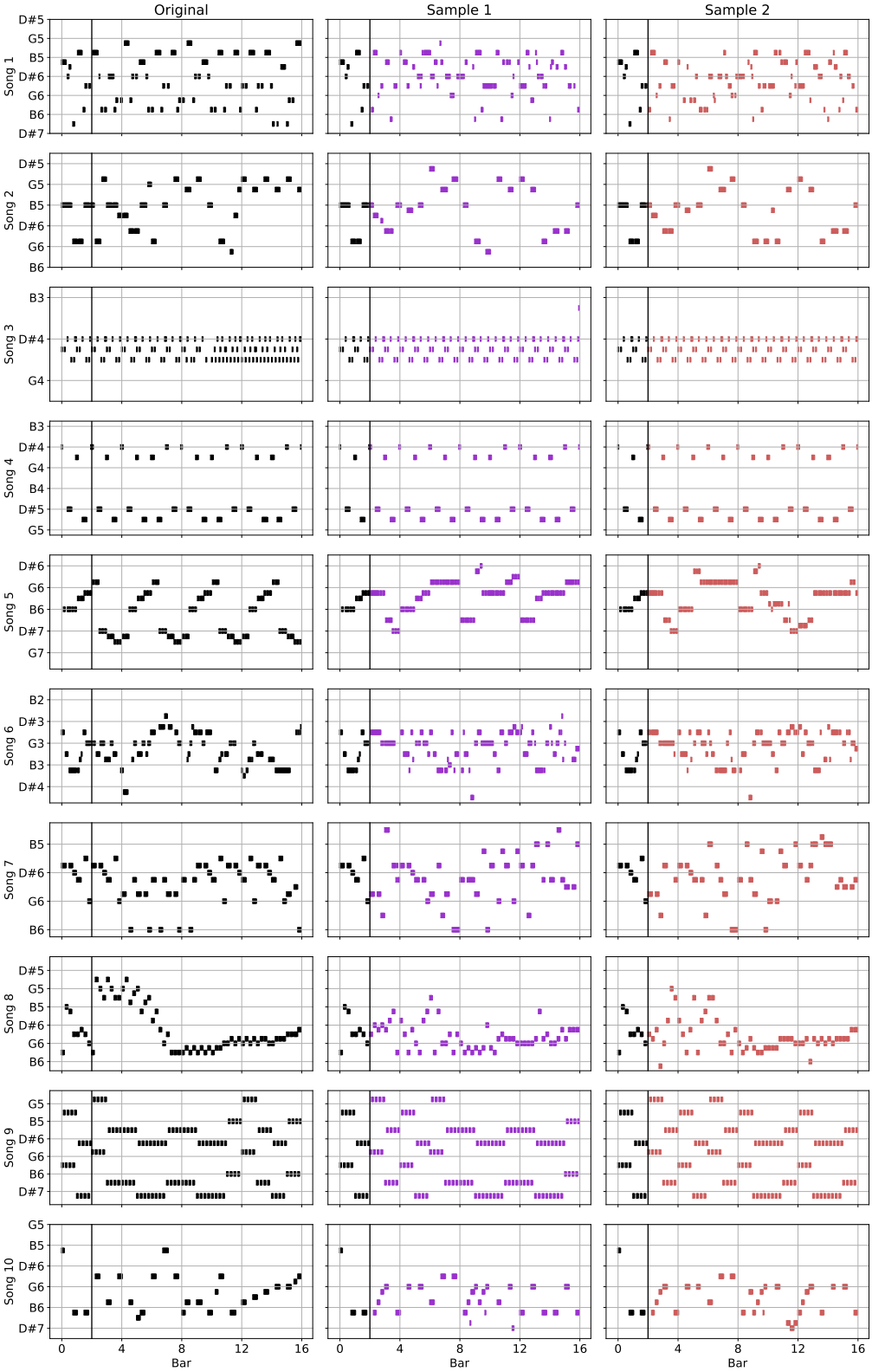


Figure 13: Two conditional samples from the τLDR-0 model for each of the first 10 songs in the test dataset.

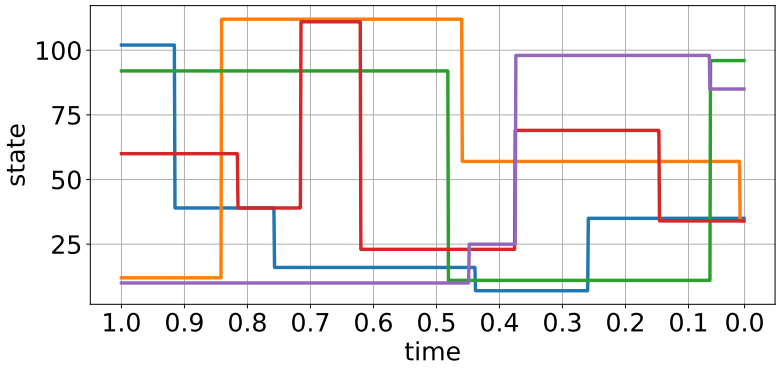


Figure 14: The progression of a selection of dimensions during the reverse sampling process for the uniform rate matrix.

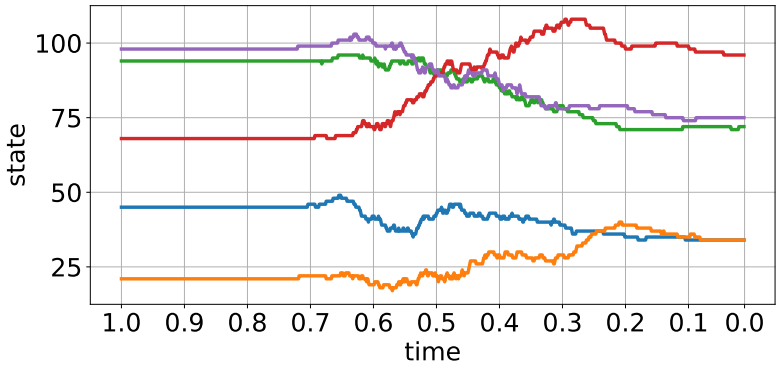


Figure 15: The progression of a selection of dimensions during the reverse sampling process for the birth/death rate matrix.

for learning from unstructured data and can have negative social impacts when misused. For example, they can be used to spread misinformation by reducing the resources required to create realistic fake content. Furthermore, generative models will produce samples that accurately reflect the statistics of their training dataset. Therefore, if samples from these models are interpreted as an objective truth without fully considering the biases present in the original data, then they can perpetuate discrimination against minority groups.

In this work, we train on datasets that contain less sensitive data such as pictures of objects and music samples. The methods we presented, however, could be used to model images of people or text from the internet which will contain biases and potentially harmful content that the model will then learn from and reproduce. Great care must be taken when training these models on real world datasets and when deploying them so as to mitigate and prevent the harms that they can cause.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	A Continuous Time Framework for Discrete Denoising Models
Publication Status	Published
Publication Details	Campbell, A., Benton, J., De Bortoli, V., Rainforth, T., Deligiannidis, G. and Doucet, A., 2022. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, *35*, pp.28266-28279

Student Confirmation

Student Name:	Andrew Campbell		
Contribution to the Paper	- Lead author - Wrote the code and ran all the experiments. - Wrote the majority of the manuscript with additions and revision from collaborators. - Wrote initial proof and denoising form of Proposition 1. Wrote the limit of discrete time ELBO proof of Proposition 2. Conceptualized and proved Proposition 3.		
Signature		Date	08-Jan-2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Arnaud Doucet, Senior Staff Research Scientist Google DeepMind			
Supervisor comments I agree with the description of the contributions.			
Signature		Date	09-Jan-2025

This completed form should be included in the thesis, at the end of the relevant chapter.

4

Trans-Dimensional Generative Modeling via Jump Diffusion Models

Trans-Dimensional Generative Modeling via Jump Diffusion Models

Andrew Campbell[1] **William Harvey**[2] **Christian Weilbach**[2]

Valentin De Bortoli[3] **Tom Rainforth**[1] **Arnaud Doucet**[1]

[1]Department of Statistics, University of Oxford, UK
[2] Department of Computer Science, University of British Columbia, Vancouver, Canada
[3]CNRS ENS Ulm, Paris, France
{campbell, rainforth, doucet}@stats.ox.ac.uk
{wsgh, weilbach}@cs.ubc.ca
valentin.debortoli@gmail.com

Abstract

We propose a new class of generative models that naturally handle data of varying dimensionality by jointly modeling the state and dimension of each datapoint. The generative process is formulated as a jump diffusion process that makes jumps between different dimensional spaces. We first define a dimension destroying forward noising process, before deriving the dimension creating time-reversed generative process along with a novel evidence lower bound training objective for learning to approximate it. Simulating our learned approximation to the time-reversed generative process then provides an effective way of sampling data of varying dimensionality by jointly generating state values and dimensions. We demonstrate our approach on molecular and video datasets of varying dimensionality, reporting better compatibility with test-time diffusion guidance imputation tasks and improved interpolation capabilities versus fixed dimensional models that generate state values and dimensions separately.

1 Introduction

Generative models based on diffusion processes [1–3] have become widely used in solving a range of problems including text-to-image generation [4, 5], audio synthesis [6] and protein design [7]. These models define a forward noising diffusion process that corrupts data to noise and then learn the corresponding time-reversed backward generative process that generates novel datapoints from noise.

In many applications, for example generating novel molecules [8] or videos [9, 10], the dimension of the data can also vary. For example, a molecule can contain a varying number of atoms and a video can contain a varying number of frames. When defining a generative model over these data-types, it is therefore necessary to model the number of dimensions along with the raw values of each of its dimensions (the state). Previous approaches to modeling such data have relied on first sampling the number of dimensions from the empirical distribution obtained from the training data, and then sampling data using a fixed dimension diffusion model (FDDM) conditioned on this number of dimensions [8]. For conditional modeling, where the number of dimensions may depend on the observations, this approach does not apply and we are forced to first train an auxiliary model that predicts the number of dimensions given the observations [11].

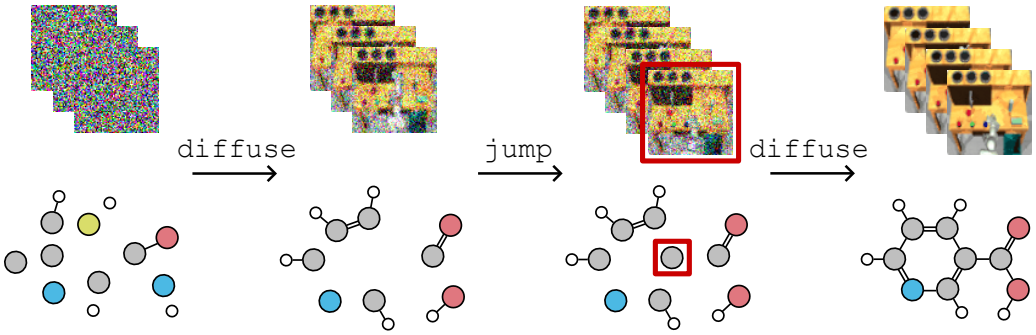


Figure 1: Illustration of the jump diffusion generative process on videos and molecules. The generative process consists of two parts: a diffusion part which denoises the current set of frames/atoms and a jump part which adds on a suitable number of new frames/atoms such that the final generation is a clean synthetic datapoint of an appropriate size.

This approach to trans-dimensional generative modeling is fundamentally limited due to the complete separation of dimension generation and state value generation. This is exemplified in the common use case of conditional diffusion guidance. Here, an unconditional diffusion model is trained that end-users can then easily and cheaply condition on their task of interest through guiding the generative diffusion process [3, 12–14] without needing to perform any further training or fine-tuning of the model on their task of interest. Since the diffusion occurs in a fixed dimensional space, there is no way for the guidance to appropriately guide the dimension of the generated datapoint. This can lead to incorrect generations for datasets where the dimension greatly affects the nature of the datapoint created, e.g. small molecules have completely different properties to large molecules.

To generate data of varying dimensionality, we propose a jump diffusion based generative model that jointly generates both the dimension and the state. Our model can be seen as a unification of diffusion models which generate all dimensions in parallel with autoregressive type models which generate dimensions sequentially. We derive the model through constructing a forward noising process that adds noise and removes dimensions and a backward generative process that denoises and adds dimensions, see Figure 1. We derive the optimum backward generative process as the time-reversal of the forward noising process and derive a novel learning objective to learn this backward process from data. We demonstrate the advantages of our method on molecular and video datasets finding our method achieves superior guided generation performance and produces more representative data interpolations across dimensions.

2 Background

Standard continuous-time diffusion models [3, 15–17] define a forward diffusion process through a stochastic differential equation (SDE) where $\mathbf{x}_0 \sim p_{\text{data}}$ and, for $t > 0$,

$$\mathrm{d}\mathbf{x}_t = \overrightarrow{\mathbf{b}}_t(\mathbf{x}_t)\mathrm{d}t + g_t \mathrm{d}\mathbf{w}_t, \tag{1}$$

where $\mathbf{x}_t \in \mathbb{R}^d$ is the current state, $\overrightarrow{\mathbf{b}}_t : \mathbb{R}^d \to \mathbb{R}^d$ is the drift and $g_t \in \mathbb{R}$ is the diffusion coefficient. $\mathrm{d}\mathbf{w}_t$ is a Brownian motion increment on $\mathbb{R}^d$. This SDE can be understood intuitively by noting that in each infinitesimal timestep, we move slightly in the direction of the drift $\overrightarrow{\mathbf{b}}_t$ and inject a small amount of Gaussian noised governed by g_t. Let $p_t(\mathbf{x}_t)$ denote the distribution of $\mathbf{x}_t$ for the forward diffusion process (1) so that $p_0(\mathbf{x}_0) = p_{\text{data}}(\mathbf{x}_0)$. $\overrightarrow{\mathbf{b}}_t$ and g_t are set such that at time $t = T$, $p_T(\mathbf{x}_T)$ is close to $p_{\text{ref}}(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; 0, I_d)$; e.g. $\overrightarrow{\mathbf{b}}_t(\mathbf{x}_t) = -\frac{1}{2}\beta_t \mathbf{x}_t$, $g_t = \sqrt{\beta_t}$ for $\beta_t > 0$ [2, 3].

The time-reversal of the forward diffusion (1) is also a diffusion [18, 19] which runs backwards in time from $p_T(\mathbf{x}_T)$ to $p_0(\mathbf{x}_0)$ and satisfies the following reverse time SDE

$$\mathrm{d}\mathbf{x}_t = \overleftarrow{\mathbf{b}}_t(\mathbf{x}_t)\mathrm{d}t + g_t \mathrm{d}\hat{\mathbf{w}}_t,$$

where $\overleftarrow{\mathbf{b}}_t(\mathbf{x}_t) = \overrightarrow{\mathbf{b}}_t(\mathbf{x}_t) - g_t^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, $\mathrm{d}t$ is a negative infinitesimal time step and $\mathrm{d}\hat{\mathbf{w}}_t$ is a Brownian motion increment when time flows backwards. Unfortunately, both the terminal

distribution, $p_T(\mathbf{x}_T)$, and the score, $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$, are unknown in practice. A generative model is obtained by approximating p_T with p_{ref} and learning an approximation $s_t^\theta(\mathbf{x}_t)$ to $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ typically using denoising score matching [20], i.e.

$$\min_\theta \quad \mathbb{E}_{\mathcal{U}(t;0,T)p_{0,t}(\mathbf{x}_0,\mathbf{x}_t)}[\|s_t^\theta(\mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{x}_0)\|^2]. \tag{2}$$

For a flexible model class, s^θ, we get $s_t^\theta(\mathbf{x}_t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ at the minimizing parameter.

3 Trans-Dimensional Generative Model

Instead of working with fixed dimension datapoints, we will instead assume our datapoints consist of a variable number of components. A datapoint $\mathbf{X}$ consists of n components each of dimension d. For ease of notation, each datapoint will explicitly store both the number of components, n, and the state values, $\mathbf{x}$, giving $\mathbf{X} = (n, \mathbf{x})$. Since each datapoint can have a variable number of components from $n = 1$ to $n = N$, our overall space that our datapoints live in is the union of all these possibilities, $\mathbf{X} \in \mathcal{X} = \bigcup_{n=1}^N \{n\} \times \mathbb{R}^{nd}$. For example, for a varying size point cloud dataset, components would refer to points in the cloud, each containing (x, y, z) coordinates giving $d = 3$ and the maximum possible number of points in the cloud is N.

Broadly speaking, our approach will follow the same framework as previous diffusion generative models. We will first define a forward noising process that both corrupts state values with Gaussian noise and progressively deletes dimensions. We then learn an approximation to the time-reversal giving a backward generative process that simultaneously denoises whilst also progressively adding dimensions back until a synthetic datapoint of appropriate dimensionality has been constructed.

3.1 Forward Process

Our forward and backward processes will be defined through jump diffusions. A jump diffusion process has two components, the diffusion part and the jump part. Between jumps, the process evolves according to a standard SDE. When a jump occurs, the process transitions to a different dimensional space with the new value for the process being drawn from a transition kernel $K_t(\mathbf{Y}|\mathbf{X}) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$. Letting $\mathbf{Y} = (m, \mathbf{y})$, the transition kernel satisfies $\sum_m \int_{\mathbf{y}} K_t(m, \mathbf{y}|\mathbf{X})\mathrm{d}\mathbf{y} = 1$ and $\int_{\mathbf{y}} K_t(m = n, \mathbf{y}|\mathbf{X})\mathrm{d}\mathbf{y} = 0$. The rate at which jumps occur (jumps per unit time) is given by a rate function $\lambda_t(\mathbf{X}) : \mathcal{X} \to \mathbb{R}_{\geq 0}$. For an infinitesimal timestep $\mathrm{d}t$, the jump diffusion can be written as

$$\begin{aligned}
\textbf{Jump} \qquad & \mathbf{X}_t' = \begin{cases} \mathbf{X}_t & \text{with probability } 1 - \lambda_t(\mathbf{X}_t)\mathrm{d}t \\ \mathbf{Y} \sim K_t(\mathbf{Y}|\mathbf{X}_t) & \text{with probability } \lambda_t(\mathbf{X}_t)\mathrm{d}t \end{cases} \\
\textbf{Diffusion} \qquad & \mathbf{x}_{t+\mathrm{d}t} = \mathbf{x}_t' + \mathbf{b}_t(\mathbf{X}_t')\mathrm{d}t + g_t\mathrm{d}\mathbf{w}_t \qquad n_{t+\mathrm{d}t} = n_t'
\end{aligned}$$

with $\mathbf{X}_t \triangleq (n_t, \mathbf{x}_t)$ and $\mathbf{X}_{t+\mathrm{d}t} \triangleq (n_{t+\mathrm{d}t}, \mathbf{x}_{t+\mathrm{d}t})$ and $\mathrm{d}\mathbf{w}_t$ being a Brownian motion increment on $\mathbb{R}^{n_t' d}$. We provide a more formal definition in Appendix A.

With the jump diffusion formalism in hand, we can now construct our forward noising process. We will use the diffusion part to corrupt existing state values with Gaussian noise and the jump part to destroy dimensions. For the diffusion part, we use the VP-SDE introduced in [2, 3] with $\overrightarrow{\mathbf{b}}_t(\mathbf{X}) = -\frac{1}{2}\beta_t\mathbf{x}$ and $\overrightarrow{g}_t = \sqrt{\beta_t}$ with $\beta_t \geq 0$.

When a jump occurs in the forward process, one component of the current state will be deleted. For example, one point in a point cloud or a single frame in a video is deleted. The rate at which these deletions occur is set by a user-defined forward rate $\overrightarrow{\lambda}_t(\mathbf{X})$. To formalize the deletion, we need to introduce some more notation. We let $K^{\text{del}}(i|n)$ be a user-defined distribution over which component of the current state to delete. We also define $\text{del} : \mathcal{X} \times \mathbb{N} \to \mathcal{X}$ to be the deletion operator that deletes a specified component. Specifically, $(n-1, \mathbf{y}) = \text{del}((n, \mathbf{x}), i)$ where $\mathbf{y} \in \mathbb{R}^{(n-1)d}$ has the same values as $\mathbf{x} \in \mathbb{R}^{nd}$ except for the d values corresponding to the ith component which have been removed. We can now define the forward jump transition kernel as $\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) = \sum_{i=1}^n K^{\text{del}}(i|n)\delta_{\text{del}(\mathbf{X},i)}(\mathbf{Y})$. We note that only one component is ever deleted at a time meaning $\overrightarrow{K}_t(m, \mathbf{y}|\mathbf{X}) = 0$ for $m \neq n - 1$. Further, the choice of $K^{\text{del}}(i|n)$ will dictate the behaviour of the reverse generative process. If we set $K^{\text{del}}(i|n) = \mathbb{I}\{i = n\}$ then we only ever delete the final component and so in the reverse generative

Table 1: Summary of forward and parameterized backward processes

Direction	$\mathbf{b}_t$	g_t	$\lambda_t(\mathbf{X})$	$K_t(\mathbf{Y}\|\mathbf{X})$
Forward	$-\frac{1}{2}\beta_t\mathbf{x}$	$\sqrt{\beta_t}$	$\overrightarrow{\lambda}_t(n)$	$\sum_{i=1}^{n} K^{\text{del}}(i\|n)\delta_{\text{del}(\mathbf{X},i)}(\mathbf{Y})$
Backward	$-\frac{1}{2}\beta_t\mathbf{x} - s_t^\theta(\mathbf{X})$	$\sqrt{\beta_t}$	$\overleftarrow{\lambda}_t^\theta(\mathbf{X})$	$\int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t^\theta(\mathbf{y}^{\text{add}}, i\|\mathbf{X})\delta_{\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)}(\mathbf{Y})\mathrm{d}\mathbf{y}^{\text{add}}$

direction, datapoints are created additively, appending components onto the end of the current state. Alternatively, if we set $K^{\text{del}}(i|n) = 1/n$ then components are deleted uniformly at random during forward corruption and in the reverse generative process, the model will need to pick the most suitable location for a new component from all possible positions.

The forward noising process is simulated from $t = 0$ to $t = T$ and should be such that at time $t = T$, the marginal probability $p_t(\mathbf{X})$ should be close to a reference measure $p_{\text{ref}}(\mathbf{X})$ that can be sampled from. We set $p_{\text{ref}}(\mathbf{X}) = \mathbb{I}\{n = 1\}\mathcal{N}(\mathbf{x}; 0, I_d)$ where $\mathbb{I}\{n = 1\}$ is 1 when $n = 1$ and 0 otherwise. To be close to p_{ref}, for the jump part, we set $\overrightarrow{\lambda}_t$ high enough such that at time $t = T$ there is a high probability that all but one of the components in the original datapoint have been deleted. For simplicity, we also set $\overrightarrow{\lambda}_t$ to depend only on the current dimension $\overrightarrow{\lambda}_t(\mathbf{X}) = \overrightarrow{\lambda}_t(n)$ with $\overrightarrow{\lambda}_t(n = 1) = 0$ so that the forward process stops deleting components when there is only 1 left. In our experiments, we demonstrate the trade-offs between different rate schedules in time. For the diffusion part, we use the standard diffusion β_t schedule [2, 3] so that we are close to $\mathcal{N}(\mathbf{x}; 0, I_d)$.

3.2 Backward Process

The backward generative process will simultaneously denoise and add dimensions back in order to construct the final datapoint. It will consist of a backward drift $\overleftarrow{\mathbf{b}}_t(\mathbf{X})$, diffusion coefficient $\overleftarrow{g}_t$, rate $\overleftarrow{\lambda}_t(\mathbf{X})$ and transition kernel $\overleftarrow{K}_t(\mathbf{Y}|\mathbf{X})$. We would like these quantities to be such that the backward process is the time-reversal of the forward process. In order to find the time-reversal of the forward process, we must first introduce some notation to describe $\overleftarrow{K}_t(\mathbf{Y}|\mathbf{X})$. $\overleftarrow{K}_t(\mathbf{Y}|\mathbf{X})$ should undo the forward deletion operation. Since $\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})$ chooses a component and then deletes it, $\overleftarrow{K}_t(\mathbf{Y}|\mathbf{X})$ will need to generate the state values for a new component, decide where the component should be placed and then insert it at this location. Our new component will be denoted $\mathbf{y}^{\text{add}} \in \mathbb{R}^d$. The insertion operator is defined as $\text{ins} : \mathcal{X} \times \mathbb{R}^d \times \mathbb{N} \to \mathcal{X}$. It takes in the current value $\mathbf{X}$, the new component $\mathbf{y}^{\text{add}}$ and an index $i \in \{1, \dots, n+1\}$ and inserts $\mathbf{y}^{\text{add}}$ into $\mathbf{X}$ at location i such that the resulting value $\mathbf{Y} = \text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)$ has $\text{del}(\mathbf{Y}, i) = \mathbf{X}$. We denote the joint conditional distribution over the newly added component and the index at which it is inserted as $A_t(\mathbf{y}^{\text{add}}, i|\mathbf{X})$. We therefore have $\overleftarrow{K}_t(\mathbf{Y}|\mathbf{X}) = \int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t(\mathbf{y}^{\text{add}}, i|\mathbf{X})\delta_{\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)}(\mathbf{Y})\mathrm{d}\mathbf{y}^{\text{add}}$. Noting that only one component is ever added at a time, we have $\overleftarrow{K}_t(m, \mathbf{y}|\mathbf{X}) = 0$ for $m \neq n + 1$.

This backward process formalism can be seen as a unification of diffusion models with autoregressive models. The diffusion part $\overleftarrow{\mathbf{b}}_t$ denoises the current set of components in parallel, whilst the autoregressive part $A_t(\mathbf{y}^{\text{add}}, i|\mathbf{X})$ predicts a new component and its location. $\overleftarrow{\lambda}_t(\mathbf{X})$ is the glue between these parts controlling when and how many new components are added during generation.

We now give the optimum values for $\overleftarrow{\mathbf{b}}_t(\mathbf{X})$, $\overleftarrow{g}_t$, $\overleftarrow{\lambda}_t(\mathbf{X})$ and $A_t(\mathbf{y}^{\text{add}}, i|\mathbf{X})$ such that the backward process is the time-reversal of the forward process.

Proposition 1. *The time reversal of a forward jump diffusion process given by drift $\overrightarrow{\mathbf{b}}_t$, diffusion coefficient $\overrightarrow{g}_t$, rate $\overrightarrow{\lambda}_t(n)$ and transition kernel $\sum_{i=1}^{n} K^{\text{del}}(i|n)\delta_{\text{del}(\mathbf{X},i)}(\mathbf{Y})$ is given by a jump diffusion process with drift $\overleftarrow{\mathbf{b}}_t^*(\mathbf{X})$, diffusion coefficient $\overleftarrow{g}_t^*$, rate $\overleftarrow{\lambda}_t^*(\mathbf{X})$ and transition kernel*

$\int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X}) \delta_{\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)}(\mathbf{Y}) \mathrm{d}\mathbf{y}^{\text{add}}$ *as defined below*

$$\overleftarrow{\mathbf{b}}_t^*(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X}) - \nabla_{\mathbf{x}} \log p_t(\mathbf{X}), \quad \overleftarrow{g}_t^* = \overrightarrow{g}_t,$$

$$\overleftarrow{\lambda}_t^*(\mathbf{X}) = \overrightarrow{\lambda}_t(n+1) \frac{\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1) \int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)) \mathrm{d}\mathbf{y}^{\text{add}}}{p_t(\mathbf{X})},$$

$$A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X}) \propto p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)) K^{\text{del}}(i|n+1).$$

All proofs are given in Appendix A. The expressions for $\overleftarrow{\mathbf{b}}_t^*$ and $\overleftarrow{g}_t^*$ are the same as for a standard diffusion except for replacing $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ with $\nabla_{\mathbf{x}} \log p_t(\mathbf{X}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}|n)$ which is simply the score in the current dimension. The expression for $\overleftarrow{\lambda}_t^*$ can be understood intuitively by noting that the numerator in the probability ratio is the probability that at time t, given a deletion occurs, the forward process will arrive at $\mathbf{X}$. If this is higher than the raw probability at time t that the forward process is at $\mathbf{X}$ (the denominator) then we should have high $\overleftarrow{\lambda}_t^*$ because $\mathbf{X}$ is likely the result of a deletion of a larger datapoint. Finally the optimum $A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X})$ is simply the conditional distribution of $\mathbf{y}^{\text{add}}$ and i given $\mathbf{X}$ when the joint distribution over $\mathbf{y}^{\text{add}}, i, \mathbf{X}$ is given by $p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)) K^{\text{del}}(i|n+1)$.

3.3 Objective for Learning the Backward Process

The true $\overleftarrow{\mathbf{b}}_t^*$, $\overleftarrow{\lambda}_t^*$ and A_t^* are unknown so we need to learn approximations to them, $\overleftarrow{\mathbf{b}}_t^\theta$, $\overleftarrow{\lambda}_t^\theta$ and A_t^θ. Following Proposition 1, we set $\overleftarrow{\mathbf{b}}_t^\theta(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X}) - s_t^\theta(\mathbf{X})$ where $s_t^\theta(\mathbf{X})$ approximates $\nabla_{\mathbf{x}} \log p_t(\mathbf{X})$. The forward and parameterized backward processes are summarized in Table 1.

Standard diffusion models are trained using a denoising score matching loss which can be derived from maximizing an evidence lower bound on the model probability for $\mathbb{E}_{p_{\text{data}}(\mathbf{x}_0)}[\log p_0^\theta(\mathbf{x}_0)]$ [21]. We derive here an equivalent loss to learn s_t^θ, $\overleftarrow{\lambda}_t^\theta$ and A_t^θ for our jump diffusion process by leveraging the results of [17] and [22]. Before presenting this loss, we first introduce some notation. Our objective for $s_t^\theta(\mathbf{X}_t)$ will resemble denoising score matching (2) but instead involve the conditional score $\nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) = \nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0, n_t)$. This is difficult to calculate directly due to a combinatorial sum over the different ways the components of $\mathbf{X}_0$ can be deleted to get to $\mathbf{X}_t$. We avoid this problem by equivalently conditioning on a mask variable $M_t \in \{0,1\}^{n_0}$ that is 0 for components of $\mathbf{X}_0$ that have been deleted to get to $\mathbf{X}_t$ and 1 for components that remain in $\mathbf{X}_t$. This makes our denoising score matching target easy to calculate: $\nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0, n_t, M_t) = \frac{\sqrt{\alpha_t} M_t(\mathbf{x}_0) - \mathbf{x}_t}{1-\alpha_t}$ where $\alpha_t = \exp(-\int_0^t \beta(s)\mathrm{d}s)$ [3]. Here $M_t(\mathbf{x}_0)$ is the vector removing any components in $\mathbf{x}_0$ for which M_t is 0, thus $M_t(\mathbf{x}_0)$ and $\mathbf{x}_t$ have the same dimensionality. We now state our full objective.

Proposition 2. *For the backward generative jump diffusion process starting at $p_{\text{ref}}(\mathbf{X}_T)$ and finishing at $p_0^\theta(\mathbf{X}_0)$, an evidence lower bound on the model log-likelihood $\mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}}[\log p_0^\theta(\mathbf{x}_0)]$ is given by*

$$\mathcal{L}(\theta) = -\frac{T}{2} \mathbb{E}\Big[g_t^2 \| s_t^\theta(\mathbf{X}_t) - \nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0, n_t, M_t) \|^2 \Big] + \tag{3}$$

$$T \mathbb{E}\Big[-\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) + \overrightarrow{\lambda}_t(n_t) \log \overleftarrow{\lambda}_t^\theta(\mathbf{Y}) + \overrightarrow{\lambda}_t(n_t) \log A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \Big] + C, \tag{4}$$

where expectations are with respect to $\mathcal{U}(t; 0, T) p_{0,t}(\mathbf{X}_0, \mathbf{X}_t, M_t) K^{\text{del}}(i|n_t) \delta_{\text{del}(\mathbf{X}_t, i)}(\mathbf{Y})$, C is a constant term independent of θ and $\mathbf{X}_t = ins(\mathbf{Y}, \mathbf{x}_t^{\text{add}}, i)$. This evidence lower bound is equal to the log-likelihood when $\overleftarrow{\mathbf{b}}_t^\theta = \overleftarrow{\mathbf{b}}_t^*$, $\overleftarrow{\lambda}_t^\theta = \overleftarrow{\lambda}_t^*$ and $A_t^\theta = A_t^*$.

We now examine the objective to gain an intuition into the learning signal. Our first term (3) is an L_2 regression to a target that, as we have seen, is a scaled vector between $\mathbf{x}_t$ and $\sqrt{\alpha_t} M_t(\mathbf{x}_0)$. As the solution to an L_2 regression problem is the conditional expectation of the target, $s_t^\theta(\mathbf{X}_t)$ will learn to predict vectors pointing towards $\mathbf{x}_0$ averaged over the possible correspondences between dimensions of $\mathbf{x}_t$ and dimensions of $\mathbf{x}_0$. Thus, during sampling, $s_t^\theta(\mathbf{X}_t)$ provides a suitable direction to adjust the current value $\mathbf{X}_t$ taking into account the fact $\mathbf{X}_t$ represents only a noisy subpart of a clean whole $\mathbf{X}_0$.

The second term (4) gives a learning signal for $\overleftarrow{\lambda}_t^\theta$ and A_t^θ. For A_t^θ, we simply have a maximum likelihood objective, predicting the missing part of $\mathbf{X}_t$ (i.e. $\mathbf{x}_t^{\text{add}}$) given the observed part of $\mathbf{X}_t$ (i.e. $\mathbf{Y}$). The signal for $\overleftarrow{\lambda}_t^\theta$ comes from balancing two terms: $-\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ and $\overrightarrow{\lambda}_t(n_t) \log \overleftarrow{\lambda}_t^\theta(\mathbf{Y})$

which encourage the value of $\overleftarrow{\lambda}_t^\theta$ to move in opposite directions. For a new test input $\mathbf{Z}$, $\overleftarrow{\lambda}_t^\theta(\mathbf{Z})$'s value needs to trade off between the two terms by learning the relative probability between $\mathbf{Z}$ being the entirety of a genuine sample from the forward process, corresponding to the $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ term in (4), or $\mathbf{Z}$ being a substructure of a genuine sample, corresponding to the $\overleftarrow{\lambda}_t^\theta(\mathbf{Y})$ term in (4). The optimum trade-off is found exactly at the time reversal $\overleftarrow{\lambda}_t^*$ as we show in Appendix A.5.

We optimize $\mathcal{L}(\theta)$ using stochastic gradient ascent, generating minibatches by first sampling $t \sim \mathcal{U}(0, T)$, $\mathbf{X}_0 \sim p_{\text{data}}$ and then computing $\mathbf{X}_t$ from the forward process. This can be done analytically for the $\overrightarrow{\lambda}_t(n)$ functions used in our experiments. We first sample n_t by analytic integration of the dimension deletion Poisson process with time inhomogeneous rate $\overrightarrow{\lambda}_t(n)$. We then add Gaussian noise independently to each dimension under $p_{t|0}(\mathbf{x}_t|\mathbf{X}_0, n_t, M_t)$ using a randomly drawn mask variable M_t. See Appendix B for further details on the efficient evaluation of our objective.

3.4 Parameterization

$s_t^\theta(\mathbf{X}_t)$, $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ and $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ will all be parameterized by neural networks. In practice, we have a single backbone network suited to the problem of interest e.g. a Transformer [23], an EGNN [24] or a UNet [25] onto which we add prediction heads for $s_t^\theta(\mathbf{X}_t)$, $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ and $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$. $s_t^\theta(\mathbf{X}_t)$ outputs a vector in $\mathbb{R}^{n_t d}$. $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ outputs a distribution over i and mean and standard deviation statistics for a Gaussian distribution over $\mathbf{y}^{\text{add}}$. Finally, having $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) \in \mathbb{R}_{\geq 0}$ be the raw output of a neural network can cause optimization issues due to the optimum $\overleftarrow{\lambda}_t^*$ including a probability ratio which can take on very large values. Instead, we learn a component prediction network $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ that predicts the number of components in $\mathbf{X}_0$ given $\mathbf{X}_t$. To convert this into $\overleftarrow{\lambda}^\theta(\mathbf{X}_t)$, we show in Proposition 3 how the optimum $\overleftarrow{\lambda}_t^*(\mathbf{X}_t)$ is an analytic function of the true $p_{0|t}(n_0|\mathbf{X}_t)$. We then plug $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ into Proposition 3 to obtain an approximation of $\overleftarrow{\lambda}_t^*(\mathbf{X}_t)$.

Proposition 3. *We have*

$$\overleftarrow{\lambda}_t^*(\mathbf{X}_t) = \overrightarrow{\lambda}_t(n_t + 1) \sum_{n_0=1}^{N} \frac{p_{t|0}(n_t + 1|n_0)}{p_{t|0}(n_t|n_0)} p_{0|t}(n_0|\mathbf{X}_t),$$

where $\mathbf{X}_t = (n_t, \mathbf{x}_t)$ and $p_{t|0}(n_t + 1|n_0)$ and $p_{t|0}(n_t|n_0)$ are both easily calculable distributions from the forward dimension deletion process.

3.5 Sampling

To sample the generative process, we numerically integrate the learned backward jump diffusion process using time-step δt. Intuitively, it is simply the standard continuous time diffusion sampling scheme [3] but at each timestep we check whether a jump has occurred and if it has, sample the new component and insert it at the chosen index as explained by Algorithm 1.

Algorithm 1: Sampling the Generative Process

$t \leftarrow T$
$\mathbf{X} \sim p_{\text{ref}}(\mathbf{X}) = \mathbb{I}\{n = 1\}\mathcal{N}(\mathbf{x}; 0, I_d)$
while $t > 0$ **do**
  if $u < \overleftarrow{\lambda}_t^\theta(\mathbf{X})\delta t$ with $u \sim \mathcal{U}(0, 1)$ **then**
    Sample $\mathbf{x}^{\text{add}}, i \sim A_t^\theta(\mathbf{x}^{\text{add}}, i|\mathbf{X})$
    $\mathbf{X} \leftarrow \text{ins}(\mathbf{X}, \mathbf{x}^{\text{add}}, i)$
  end
  $\mathbf{x} \leftarrow \mathbf{x} - \overleftarrow{\mathbf{b}}_t^\theta(\mathbf{X})\delta t + g_t\sqrt{\delta t}\epsilon$ with $\epsilon \sim \mathcal{N}(0, I_{nd})$
  $\mathbf{X} \leftarrow (n, \mathbf{x}), t \leftarrow t - \delta t$
end

4 Related Work

Our method jointly generates both dimensions and state values during the generative process whereas prior approaches [8, 11] are forced to first sample the number of dimensions and then run the diffusion process in this fixed dimension. When diffusion guidance is applied to these unconditional models [14, 26], users need to pick by hand the number of dimensions independent of the conditioning information even though the number of dimensions can be correlated with the conditioning parameter.

Instead of automatically learning when and how many dimensions to add during the generative process, previous work focusing on images [27, 28] hand pick dimension jump points such that the

resolution of images is increased during sampling and reaches a certain pre-defined desired resolution at the end of the generative process. Further, rather than using any equivalent of A_t^θ, the values for new dimensions are simply filled in with Gaussian noise. These approaches mainly focus on efficiency rather than flexible generation as we do here.

The first term in our learning objective in Proposition 2 corresponds to learning the continuous part of our process (the diffusion) and the second corresponds to learning the discrete part of our process (the jumps). The first term can be seen as a trans-dimensional extension of standard denoising score matching [20] whilst the second bears similarity to the discrete space ELBO derived in [29].

Finally, jump diffusions also have a long history of use in Bayesian inference, where one aims to draw samples from a trans-dimensional target posterior distribution based on an unnormalized version of its density [30]: an ergodic jump diffusion is designed which admits the target as the invariant distribution [30–32]. The invariant distribution is not preserved when time-discretizing the process. However, it was shown in [33, 34] how general jump proposals could be built and how this process could be "Metropolized" to obtain a discrete-time Markov process admitting the correct invariant distribution, yielding the popular Reversible Jump Markov Chain Monte Carlo algorithm. Our setup differs significantly as we only have access to samples in the form of data, not an unnormalized target.

5 Experiments

5.1 Molecules

We now show how our model provides significant benefits for diffusion guidance and interpolation tasks. We model the QM9 dataset [35, 36] of 100K varying size molecules. Following [8], we consider each molecule as a 3-dimensional point cloud of atoms, each atom having the features: (x, y, z) coordinates, a one-hot encoded atom type, and an integer charge value. Bonds are inferred from inter-atomic distances. We use an EGNN [24] backbone with three heads to predict s_t^θ, $p_{0|t}^\theta(n_0|\mathbf{X}_t)$, and A_t^θ. We uniformly delete dimensions, $K^{\text{del}}(i|n) = 1/n$, and since a point cloud is permutation invariant, $A_t^\theta(\mathbf{y}^{\text{add}}|\mathbf{X}_t)$ need only predict new dimension values. We set $\overrightarrow{\lambda}_t$ to a constant except for $t < 0.1T$, where we set $\overrightarrow{\lambda}_{t<0.1T} = 0$. This ensures that all dimensions are added with enough generation time remaining for the diffusion process to finalize all state values.

We visualize sampling from our learned generative process in Figure 2; note how the process jointly creates a suitable number of atoms whilst adjusting their positions and identities. Before moving on to apply diffusion guidance which is the focus of our experiments, we first verify our unconditional sample quality in Table 2 and find we perform comparably to the results reported in [8] which use an FDDM. We ablate our choice of $\overrightarrow{\lambda}_t$ by comparing with setting $\overrightarrow{\lambda}_t$ to a constant for all t and with setting $\overrightarrow{\lambda}_t = 0$ for $t < 0.9T$ (rather than just for $t < 0.1T$). We find that the constant $\overrightarrow{\lambda}_t$ performs worse due to the occasional component being added late in the generation process without enough time for the diffusion process to finalize its value. We find the $\overrightarrow{\lambda}_{t<0.9T} = 0$ setting to have satisfactory sample quality however this choice of $\overrightarrow{\lambda}_t$ introduces issues during diffusion guided generation as we see next. Finally, we ablate the parameterization of Proposition 3 by learning $\overrightarrow{\lambda}_t^\theta(\mathbf{X}_t) \in \mathbb{R}$ directly as the output of a neural network head. We find that this reduces sample quality due to the more well-behaved nature of the target, $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ when using Proposition 3. We note pure autoregressive models perform significantly worse than diffusion based models as found in [8].

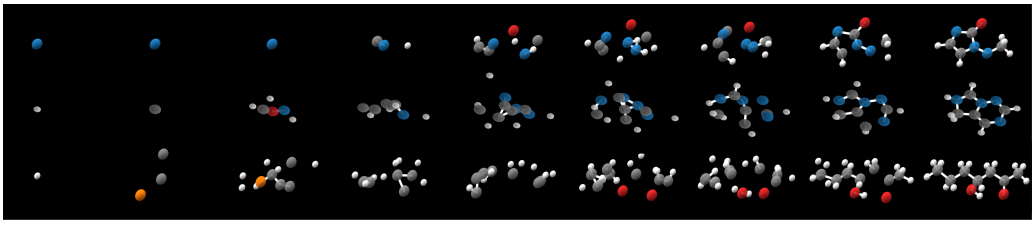


Figure 2: Visualization of the jump-diffusion backward generative process on molecules.

Table 2: Sample quality metrics for unconditional molecule generation. An atom is stable if it has the correct valency whilst a molecule is considered stable if all of its atoms are stable. Molecular validity is measured using RDKit [37]. All methods use 1000 simulation steps and draw 10000 samples.

Method	% Atom Stable (↑)	% Molecule Stable (↑)	% Valid (↑)
FDDM [8]	**98.7**	82.0	91.9
TDDM (ours)	98.3	**87.2**	**92.3**
TDDM, const $\overrightarrow{\lambda}_t$	96.7	79.1	86.7
TDDM, $\overrightarrow{\lambda}_{t<0.9T} = 0$	97.7	82.6	89.4
TDDM w/o Prop. 3	97.0	66.9	87.1

Table 3: Conditional Molecule Generation for 10 conditioning tasks that each result in a different dimension distribution. We report dimension error as the average Hellinger distance between the generated and ground truth dimension distributions for that property as well as average sample quality metrics. Standard deviations are given across the 10 conditioning tasks. We report in bold values that are statistically indistinguishable from the best result at the 5% level using a two-sided Wilcoxon signed rank test across the 10 conditioning tasks.

Method	Dimension Error (↓)	% Atom Stable (↑)	% Molecule Stable (↑)	% Valid (↑)
FDDM	$0.511_{\pm 0.19}$	$93.5_{\pm 1.1}$	$31.3_{\pm 6.3}$	$65.2_{\pm 10.3}$
TDDM	$\mathbf{0.134_{\pm 0.076}}$	$93.5_{\pm 2.6}$	$\mathbf{59.1_{\pm 11}}$	$\mathbf{74.8_{\pm 9.3}}$
TDDM, const $\overrightarrow{\lambda}_t$	$0.226_{\pm 0.17}$	$88.9_{\pm 4.8}$	$43.6_{\pm 15}$	$63.4_{\pm 14}$
TDDM, $\overrightarrow{\lambda}_{t<0.9T} = 0$	$0.390_{\pm 0.38}$	$\mathbf{95.0_{\pm 2.1}}$	$\mathbf{61.7_{\pm 17}}$	$\mathbf{77.8_{\pm 13}}$
TDDM w/o Prop. 3	$0.219_{\pm 0.12}$	$\mathbf{93.8_{\pm 3.2}}$	$55.0_{\pm 19}$	$73.8_{\pm 13}$

5.1.1 Trans-Dimensional Diffusion Guidance

We now apply diffusion guidance to our unconditional model in order to generate molecules that contain a certain number of desired atom types, e.g. 3 carbons or 1 oxygen and 2 nitrogens. The distribution of molecule sizes changes depending on these conditions. We generate molecules conditioned on these properties by using the reconstruction guided sampling approach introduced in [9]. This method augments the score $s_t^\theta(\mathbf{X}_t)$ such that it approximates $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{X}_t|y)$ rather than $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{X}_t)$ (where y is the conditioning information) by adding on a term approximating $\nabla_{\mathbf{x}_t} \log p_t(y|\mathbf{X}_t)$ with $p_t(y|\mathbf{X}_t) = \sum_{n_0} \int_{\mathbf{x}_0} p(y|\mathbf{X}_0) p_{0|t}(\mathbf{X}_0|\mathbf{X}_t) \mathrm{d}\mathbf{x}_0$. This guides $\mathbf{x}_t$ such that it is consistent with y. Since $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ has access to $\mathbf{x}_t$, it will cause n_t to automatically also be consistent with y without the user needing to input any information on how the conditioning information relates to the size of the datapoints. We give further details on diffusion guidance in Appendix C.

We show our results in Table 3. In order to perform guidance on the FDDM baseline, we implement the model from [8] in continuous time and initialize the dimension from the empirically observed dimension distribution in the dataset. This accounts for the case of an end user attempting to guide a unconditional model with access to no further information. We find that TDDM produces samples whose dimensions much more accurately reflect the true conditional distribution of dimensions given the conditioning information. The $\overrightarrow{\lambda}_{t<0.9T} = 0$ ablation on the other hand only marginally improves the dimension error over FDDM because all dimensions are added in the generative process at a time when $\mathbf{X}_t$ is noisy and has little relation to the conditioning information. This highlights the necessity of allowing dimensions to be added throughout the generative process to gain the trans-dimensional diffusion guidance ability. The ablation with constant $\overrightarrow{\lambda}_t$ has increased dimension error over TDDM as we find that when $\overrightarrow{\lambda}_t > 0$ for all t, $\overleftarrow{\lambda}_t^\theta$ can become very large when t is close to 0 when the model has perceived a lack of dimensions. This occasionally results in too many dimensions being added hence an increased dimension error. Not using the Proposition 3 parameterization also increases dimension error due to the increased difficulty in learning $\overleftarrow{\lambda}_t^\theta$.

5.1.2 Trans-Dimensional Interpolation

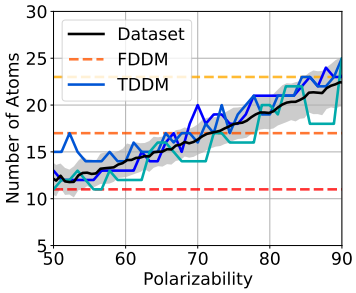


Figure 3: Number of atoms versus polarizability for 3 interpolations with fixed random noise. The dataset mean and standard deviation for the number of atoms is also shown. FDDM interpolates entirely in a fixed dimensional space hence the number of atoms is fixed for all polarizabilities.

Interpolations are a unique way of gaining insights into the effect of some conditioning parameter on a dataset of interest. To create an interpolation, a conditional generative model is first trained and then sampled with a sweep of the conditioning parameter but using fixed random noise [8]. The resulting series of synthetic datapoints share similar features due to the fixed random noise but vary in ways that are very informative as to the effect of the conditioning parameter. Attempting to interpolate with an FDDM is fundamentally limited because the entire interpolation occurs in the same dimension which is unrealistic when the conditioning parameter is heavily correlated with the dimension of the datapoint. We demonstrate this by following the setup of [8] who train a conditional FDDM conditioned on polarizability. Polarizability is the ability of a molecule's electron cloud to distort in response to an external electric field [38] with larger molecules tending to have higher polarizability. To enable us to perform a trans-dimensional interpolation, we also train a conditional version of our model conditioned on polarizability. An example interpolation with this model is shown in Figure 4. We find that indeed the size of the molecule increases with increasing polarizability, with some molecular substructures e.g. rings, being maintained across dimensions. We show how the dimension changes with polarizability during 3 interpolations in Figure 3. We find that these match the true dataset statistics much more accurately than interpolations using FDDM which first pick a dimension and carry out the entire interpolation in that fixed dimension.

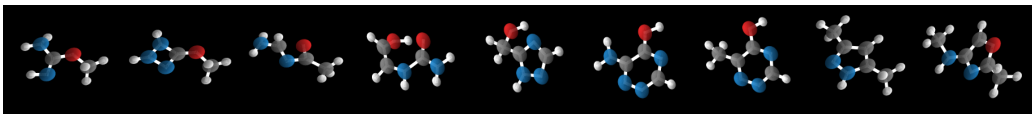


Figure 4: Sequence of generations for linearly increasing polarizability from 39 Bohr^3 to 66 Bohr^3 with fixed random noise. Note how molecular size generally increases with polarizability and how some molecular substructures are maintained between sequential generations of differing dimension. For example, between molecules 6 and 7, the single change is a nitrogen (blue) to a carbon (gray) and an extra hydrogen (white) is added to maintain the correct valency.

5.2 Video

We finally demonstrate our model on a video modeling task. Specifically we model the RoboDesk dataset [39], a video benchmark to measure the applicability of video models for planning and control problems. The videos are renderings of a robotic arm [40] performing a variety of different tasks including opening drawers and moving objects. We first train an unconditional model on videos of varying length and then perform planning by applying diffusion guidance to generate videos conditioned on an initial starting frame and a final goal frame [41]. The planning problem is then reduced to "filling in" the frames in between. Our trans-dimensional model automatically varies the number of in-filled frames during generation so that the final length of video matches the length of time the task should take, whereas the fixed dimension model relies on the unrealistic assumption that the length of time the task should take is known before generation.

We model videos at 32×32 resolution and with varying length from 2 to 35 frames. For the network backbone, we use a UNet adapted for video [42]. In contrast to molecular point clouds, our data is no longer permutation invariant hence $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ includes a prediction over the location to insert the new frame. Full experimental details are provided in Appendix D. We evaluate our approach on three planning tasks, holding stationary, sliding a door and pushing an object. An example generation conditioned on the first and last frame for the slide door task is shown in Figure 5, with

the model in-filling a plausible trajectory. We quantify our model's ability to generate videos of a length appropriate to the task in Table 4 finding on all three tasks we generate a more accurate length of video than FDDM which is forced to sample video lengths from the unconditional empirically observed length distribution in the training dataset.

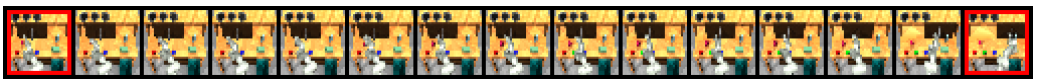


Figure 5: A sample for the slide door task conditioned on the first and last frame (highlighted).

Table 4: Dimension prediction mean absolute error for three planning tasks with standard deviations estimated over 45 samples.

Method	Stationary (↓)	Slide Door (↓)	Push Object (↓)	Average (↓)
FDDM	$14.16_{\pm 1.41}$	$13.39_{\pm 1.34}$	$17.06_{\pm 1.47}$	14.87
TDDM	$\mathbf{9.70}_{\pm \mathbf{0.99}}$	$\mathbf{11.47}_{\pm \mathbf{0.74}}$	$\mathbf{15.43}_{\pm \mathbf{0.90}}$	**12.2**

6 Discussion

In this work, we highlighted the pitfalls of performing generative modeling on varying dimensional datasets when treating state values and dimensions completely separately. We instead proposed a trans-dimensional generative model that generates both state values and dimensions jointly during the generative process. We detailed how this process can be formalized with the time-reversal of a jump diffusion and derived a novel evidence lower bound training objective for learning the generative process from data. In our experiments, we found our trans-dimensional model to provide significantly better dimension generation performance for diffusion guidance and interpolations when conditioning on properties that are heavily correlated with the dimension of a datapoint. We believe our approach can further enable generative models to be applied in a wider variety of domains where previous restrictive fixed dimension assumptions have been unsuitable.

7 Acknowledgements

The authors are grateful to Martin Buttenschoen for helpful discussions. AC acknowledges support from the EPSRC CDT in Modern Statistics and Statistical Machine Learning (EP/S023151/1). AD acknowledges support of the UK Dstl and EPSRC grant EP/R013616/1. This is part of the collaboration between US DOD, UK MOD and UK EPSRC under the Multidisciplinary University Research Initiative. He also acknowledges support from the EPSRC grants CoSines (EP/R034710/1) and Bayes4Health (EP/R018561/1). WH and CW acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program. This material is based upon work supported by the United States Air Force Research Laboratory (AFRL) under the Defense Advanced Research Projects Agency (DARPA) Data Driven Discovery Models (D3M) program (Contract No. FA8750-19-2-0222) and Learning with Less Labels (LwLL) program (Contract No.FA8750-19-C-0515). Additional support was provided by UBC's Composites Research Network (CRN), Data Science Institute (DSI) and Support for Teams to Advance Interdisciplinary Research (STAIR) Grants. This research was enabled in part by technical support and computational resources provided by WestGrid (`https://www.westgrid.ca/`) and Compute Canada (`www.computecanada.ca`). The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. `http://dx.doi.org/10.5281/zenodo.22558`

References

[1] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning*, 2015.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020.

[3] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.

[4] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, and Mohammad Fleet, David J aand Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 2022.

[5] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[6] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *International Conference on Learning Representations*, 2021.

[7] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, 2022.

[8] Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. *International Conference on Machine Learning*, 2022.

[9] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 2022.

[10] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, and Tim Saliman. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[11] Ilia Igashov, Hannes Stärk, Clément Vignac, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, and Bruno Correia. Equivariant 3d-conditional diffusion models for molecular linker design. *arXiv preprint arXiv:2210.05274*, 2022.

[12] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. *Advances in Neural Information Processing Systems*, 2021.

[13] Katherine Crowson. Clip guided diffusion. *Web Demo, https://huggingface.co/spaces/EleutherAI/clip-guided-diffusion*, 2021.

[14] Hengtong Zhang and Tingyang Xu. Towards controllable diffusion models via reward-guided exploration. *arXiv preprint arXiv:2304.07132*, 2023.

[15] Chin-Wei Huang, Jae Hyun Lim, and Aaron C Courville. A variational perspective on diffusion-based generative models and score matching. *Advances in Neural Information Processing Systems*, 2021.

[16] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 2022.

[17] Joe Benton, Yuyang Shi, Valentin De Bortoli, George Deligiannidis, and Arnaud Doucet. From denoising diffusions to denoising Markov models. *arXiv preprint arXiv:2211.03595*, 2022.

[18] Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 1982.

[19] Ulrich G Haussmann and Etienne Pardoux. Time reversal of diffusions. *The Annals of Probability*, 1986.

[20] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 2011.

[21] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 2021.

[22] Patrick Cheridito, Damir Filipović, and Marc Yor. Equivalent and absolutely continuous measure changes for jump-diffusion processes. *Annals of applied probability*, 2005.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[24] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. *International Conference on Machine Learning*, 2021.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, 2015.

[26] Tomer Weiss, Luca Cosmo, Eduardo Mayo Yanes, Sabyasachi Chakraborty, Alex M Bronstein, and Renana Gershoni-Poranne. Guided diffusion for inverse molecular design. *ChemrXiv*, 2023.

[27] Bowen Jing, Gabriele Corso, Renato Berlinghieri, and Tommi Jaakkola. Subspace diffusion generative models. *European Conference on Computer Vision*, 2022.

[28] Han Zhang, Ruili Feng, Zhantao Yang, Lianghua Huang, Yu Liu, Yifei Zhang, Yujun Shen, Deli Zhao, Jingren Zhou, and Fan Cheng. Dimensionality-varying diffusion process. *arXiv preprint arXiv:2211.16032*, 2022.

[29] Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 2022.

[30] Ulf Grenander and Michael I Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1994.

[31] David B Phillips and Adrian F M Smith. Bayesian model comparison via jump diffusions. *Markov Chain Monte Carlo in Practice*, 1995.

[32] Michael I Miller, Ulf Grenander, Joseph A O'Sullivan, and Donald L Snyder. Automatic target recognition organized via jump-diffusion algorithms. *IEEE Transactions on Image Processing*, 1997.

[33] Peter J Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 1995.

[34] Peter J Green. Trans-dimensional Markov chain Monte Carlo. *Highly Structured Stochastic Systems*, 2003.

[35] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 2012.

[36] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 2014.

[37] Rdkit: Open-source cheminformatics; `http://www.rdkit.org`. Accessed 2023.

[38] Eric V Anslyn and Dennis A Dougherty. Modern physical organic chemistry. *University Science Books*, 2005.

[39] Stephen Tian, Chelsea Finn, and Jiajun Wu. A control-centric benchmark for video prediction. *arXiv preprint arXiv:2304.13723*, 2023.

[40] Harini Kannan, Danijar Hafner, Chelsea Finn, and Dumitru Erhan. Robodesk: A multi-task reinforcement learning benchmark. *https://github.com/google-research/robodesk*, 2021.

[41] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *International Conference on Machine Learning*, 2022.

[42] William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Weilbach, and Frank Wood. Flexible diffusion modeling of long videos. *Advances in Neural Information Processing Systems*, 2022.

[43] John L Kelley. *General Topology*. Courier Dover Publications, 2017.

[44] Stewart N Ethier and Thomas G Kurtz. Markov processes: Characterization and convergence. *John Wiley & Sons*, 2009.

[45] Giovanni Conforti and Christian Léonard. Time reversal of Markov processes with jumps under a finite entropy condition. *Stochastic Processes and their Applications*, 2022.

[46] Clement Vignac and Pascal Frossard. Top-n: Equivariant set and graph generation without exchangeability. *International Conference on Learning Representations*, 2022.

[47] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019.

Appendix

This appendix is organized as follows. In Section A, we present proofs for all of our propositions. Section A.1 presents a rigorous definition of our forward process using a more specific notation. This is then used in Section A.2.1 to prove the time reversal for our jump diffusions. We also present an intuitive proof of the time reversal using notation from the main text in Section A.2.2. In Section A.3 we prove Proposition 2 using the notation from the main text. We prove Proposition 3 in Section A.4 and we analyse the optimum of our objective directly without using stochastic process theory in Section A.5. In Section B we give more details on our objective and in Section C we detail how we apply diffusion guidance to our model. We give the full details for our experiments in Section D and finally, in Section E, we discuss the broader impacts of our work.

A Proofs

A.1 Notation and Setup

We here introduce a more rigorous notation for defining our trans-dimensional notation that will be used in a rigorous proof for the time-reversal of our jump diffusion. First, while it makes sense from a methodological and experimental point of view to present our setting as a *transdimensional* one, we slightly change the point of view in order to derive our theoretical results. We extend the space $\mathbb{R}^d$ to $\hat{\mathbb{R}}^d = \mathbb{R}^d \cup \{\infty\}$ using the *one-point compactification* of the space. We refer to [43] for details on this space. The point ∞ will be understood as a mask. For instance, let $x_1, x_2, x_3 \in \mathbb{R}^d$. Then $X = (x_1, x_2, x_3) \in (\hat{\mathbb{R}}^d)^N$ with $N = 3$ corresponds to a vector for which all components are *observed* whereas $X' = (x_1, \infty, x_3) \in (\hat{\mathbb{R}}^d)^N$ corresponds to a vector for which only the components on the first and third dimension are observed. The second dimension is *masked* in that case. Doing so, we will consider diffusion models on the space $\mathsf{X} = (\hat{\mathbb{R}}^d)^N$ with $d, N \in \mathbb{N}$. In the case of a video diffusion model, N can be seen as the max number of frames. We will always consider that this space is equipped with its Borelian sigma-field $\mathcal{X}$ and all probability measures will be defined on $\mathcal{X}$.

We denote $\dim : \mathsf{X} \to \{0,1\}^N$ which is given for any $X = \{x_i\}_{i=1}^N \in \mathsf{X}$ by

$$\dim(X) = \{\delta_{\mathbb{R}^d}(x_i)\}_{i=1}^N.$$

In other words, $\dim(X)$ is a binary vector identifying the "dimension" of the vector X, i.e. which frames are observed. Going back to our example $X = (x_1, x_2, x_3) \in (\hat{\mathbb{R}}^d)^N$ and $X' = (x_1, \infty, x_3) \in (\hat{\mathbb{R}}^d)^N$, we have that $\dim(X) = \{1, 1, 1\}$ and $\dim(X') = \{1, 0, 1\}$. For any vector $u \in \{0,1\}^N$ we denote $|u| = \sum_{i=1}^N u_i$, i.e. the *active dimensions* of u (or equivalently the non-masked frames). For any $X \in \mathsf{X}$ and $\mathsf{D} \in \{0,1\}^N$, we denote $X_{\mathsf{D}} = \{X'_i\}_{i=1}^N$ with $X'_i = X_i$ if $\mathsf{D}_i = 1$ and $X'_i = \infty$ if $\mathsf{D}_i = 0$.

We denote $\mathrm{C}_b^k(\mathbb{R}^d, \mathbb{R})$ the set of functions which are k differentiable and bounded. Similarly, we denote $\mathrm{C}_b^k(\mathbb{R}^d, \mathbb{R})$ the set of functions which are k differentiable and compactly supported. The set $\mathrm{C}_0^k(\mathbb{R}^d, \mathbb{R})$ denotes the functions which are k differentiable and vanish when $\|x\| \to +\infty$. We note that $f \in \mathrm{C}(\hat{\mathbb{R}}^d)$, if $f \in \mathrm{C}(\mathbb{R}^d)$ and $f - f(\infty) \in \mathrm{C}_0(\mathbb{R}^d)$ and that $f \in \mathrm{C}^k(\hat{\mathbb{R}}^d)$ for any $k \in \mathbb{N}$ if the restriction of f to $\mathbb{R}^d$ is in $\mathrm{C}^k(\mathbb{R}^d)$ and $f \in \mathrm{C}(\hat{\mathbb{R}}^d)$.

A.1.1 Transdimensional infinitesimal generator

To introduce rigorously the *transdimensional* diffusion model defined in Section 3.1, we will introduce its *infinitesimal generator*. The infinitesimal generator of a stochastic process can be roughly defined as its "probabilistic derivative". More precisely, assume that a stochastic process $(\mathbf{X}_t)_{t\geq 0}$ admits a transition semigroup $(\mathrm{P}_t)_{t\geq 0}$, i.e. for any $t \geq 0$, $\mathsf{A} \in \mathcal{X}$ and $X \in \mathsf{X}$ we have $\mathbb{P}(\mathbf{X}_t \in \mathsf{A} \mid \mathbf{X}_0 = x) = \mathrm{P}_t(x, \mathsf{A})$, then the infinitesimal generator is defined as $\mathcal{A}(f) = \lim_{t\to 0}(\mathrm{P}_t(f) - f)/t$, for every f for which this quantity is well-defined.

Here, we start by introducing the infinitesimal generator of interest and give some intuition about its form. Then, we prove a time-reversal formula for this infinitesimal generator.

We consider $b:\ \mathbb{R}^d \to \mathbb{R}^d$, $\alpha:\ \{0,1\}^{NM} \to \mathbb{R}_+$. For any $f \in \mathrm{C}^2(\mathsf{X})$ and $X \in \mathsf{X}$ we define

$$\begin{aligned}\mathcal{A}(f)(X) = &\sum_{i=1}^N \{\langle b(X_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)\}\delta_{\mathbb{R}^d}(X_i) \\ &- \sum_{\mathsf{D}_1 \subset \mathsf{D}_0^{\Delta_0}} \cdots \sum_{\mathsf{D}_M \subset \mathsf{D}_{M-1}^{\Delta_{M-1}}} \alpha(\mathsf{D}_0, \ldots, \mathsf{D}_M) \sum_{i=0}^{M-1} (f(X) - f(X_{\mathsf{D}_{i+1}}))\delta_{\mathsf{D}_i}(\dim(X)),\end{aligned} \tag{5}$$

where $M \in \mathbb{N}$, $\mathsf{D}_0 = \{1\}^N$, $\{\Delta_j\}_{j=0}^{M-1} \in \mathbb{N}^M$ such that $\sum_{j=0}^{M-1} \Delta_j < N$ and for any $j \in \{0, \ldots, M-1\}$, $\mathsf{D}_j^{\Delta_j}$ is the subset of $\{0,1\}^{\{1,\ldots,N\}}$ such that $\mathsf{D}_{j+1} \in \mathsf{D}_j^{\Delta_j}$ if and only if $\mathsf{D}_j \cdot \mathsf{D}_{j+1} = \mathsf{D}_{j+1}$, where $\cdot$ is the pointwise multiplication operator, and $|\mathsf{D}_j| = |\mathsf{D}_{j+1}| + \Delta_j$. The condition $\mathsf{D}_j \cdot \mathsf{D}_{j+1} = \mathsf{D}_{j+1}$ means that the non-masked dimensions in D_{j+1} are also non-masked dimensions in D_j. The condition $|\mathsf{D}_j| = |\mathsf{D}_{j+1}| + \Delta_j$ means that in order to go from D_j to D_{j+1}, one needs to mask exactly Δ_j dimensions.

Therefore, a sequence $\{\Delta_j\}_{j=0}^{M-1} \in \mathbb{N}^M$ such that $\sum_{j=0}^{M-1} \Delta_j < N$ can be interpreted as a sequence of *drops* in dimension. At the core level, we have that $|\mathsf{D}_M| = N - \sum_{j=0}^{M-1} \Delta_j$. For instance if $|\mathsf{D}_M| = 1$, we have that at the end of the process, only one dimension is considered.

We choose α such that $\sum_{\mathsf{D}_1 \subset \mathsf{D}_0^{\Delta_0}} \cdots \sum_{\mathsf{D}_M \subset \mathsf{D}_{M-1}^{\Delta_{M-1}}} \alpha(\mathsf{D}_0, \ldots, \mathsf{D}_M) = 1$. Therefore, $\alpha(\mathsf{D}_0, \ldots, \mathsf{D}_M)$ corresponds to the probability to choose the *dimension path* $\mathsf{D}_0 \to \cdots \to \mathsf{D}_M$.

The part $X \mapsto \langle b(X_i), \nabla_{x_i} f(X)\rangle + \frac{1}{2}\Delta_{x_i} f(X)$ is more classical and corresponds to the *continuous part* of the diffusion process. We refer to [44] for a thorough introduction on infinitesimal generators. For simplicity, we omit the schedule coefficients in (5).

A.1.2 Justification of the form of the infinitesimal generator

For any *dimension path* $\mathsf{P} = \mathsf{D}_0 \to \cdots \to \mathsf{D}_M$ (recall that $\mathsf{D}_0 = \{1\}^N$), we define the *jump kernel* $\mathbb{J}^{\mathsf{P}}$ as follows. For any $x \in \mathsf{X}$, we have $\mathbb{J}^{\mathsf{P}}(X, \mathrm{d}Y) = \sum_{i=0}^{M-1} \delta_{\mathsf{D}_i}(\dim(X))\delta_{X_{\mathsf{D}_{i+1}}}(\mathrm{d}Y)$. This operator corresponds to the *deletion* operator introduced in Section 3.1 . Hence, for any *dimension path* $\mathsf{P} = \mathsf{D}_0 \to \cdots \to \mathsf{D}_M$, we can define the associated infinitesimal generator: for any $f \in \mathrm{C}^2(\mathsf{X})$ and $X \in \mathsf{X}$ we define

$$\mathcal{A}^{\mathsf{P}}(f)(X) = \sum_{i=1}^N \{\langle b(x_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)\}\delta_{\mathbb{R}^d}(X_i) + \int_{\mathsf{X}} (f(Y) - f(X))\mathbb{J}^{\mathsf{P}}(X, \mathrm{d}Y).$$

We can define the following *jump kernel*

$$\mathbb{J} = \sum_{\mathsf{D}_1 \subset \mathsf{D}_0^{\Delta_0}} \cdots \sum_{\mathsf{D}_M \subset \mathsf{D}_{M-1}^{\Delta_{M-1}}} \alpha(\mathsf{D}_0, \ldots, \mathsf{D}_M)\mathbb{J}^{\mathsf{P}}.$$

This corresponds to averaging the jump kernel over the different possible dimension paths. We have that for any $f \in \mathrm{C}^2(\mathsf{X})$ and $X \in \mathsf{X}$

$$\mathcal{A}(f)(X) = \sum_{i=1}^N \{\langle b(x_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)\}\delta_{\mathbb{R}^d}(X_i) + \int_{\mathsf{X}} (f(Y) - f(X))\mathbb{J}(X, \mathrm{d}Y). \tag{6}$$

In other words, $\mathcal{A} = \sum_{\mathsf{D}_1 \subset \mathsf{D}_0^{\Delta_0}} \cdots \sum_{\mathsf{D}_M \subset \mathsf{D}_{M-1}^{\Delta_{M-1}}} \alpha(\mathsf{D}_0, \ldots, \mathsf{D}_M)\mathcal{A}^{\mathsf{P}}$.

In what follows, we assume that there exists a Markov process $(\mathbf{X}_t)_{t\geq 0}$ with infinitesimal generator $\mathcal{A}$. In order to sample from $(\mathbf{X}_t)_{t\geq 0}$, one choice is to first sample the dimension path P according to the probability α. Second sample from the Markov process associated with the infinitesimal generator $\mathcal{A}^{\mathsf{P}}$. We can approximately sample from this process using the Lie-Trotter-Kato formula [44, Corollary 6.7, p.33].

Denote $(\mathrm{P}_t)_{t\geq 0}$ the semigroup associated with $\mathcal{A}^{\mathsf{P}}$, $(\mathrm{Q}_t)_{t\geq 0}$ the semigroup associated with the continuous part of $\mathcal{A}^{\mathsf{P}}$ and $(\mathrm{J}_t)_{t\geq 0}$ the semigroup associated with the jump part of $\mathcal{A}^{\mathsf{P}}$. More precisely, we have that, $(\mathrm{Q}_t)_{t\geq 0}$ is associated with $\mathcal{A}_{\mathrm{cont}}$ such that for any $f \in \mathrm{C}^2(\mathsf{X})$ and $X \in \mathsf{X}$

$$\mathcal{A}_{\mathrm{cont}}(f)(X) = \sum_{i=1}^N \{\langle b(X_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)\}.$$

In addition, we have that, $(\mathrm{Q}_t)_{t\geq 0}$ is associated with $\mathcal{A}^{\mathsf{P}}_{\mathrm{jump}}$ such that for any $f \in \mathrm{C}^2(\mathsf{X})$ and $X \in \mathsf{X}$

$$\mathcal{A}^{\mathsf{P}}_{\mathrm{jump}}(f)(X) = \int_{\mathsf{X}} (f(Y) - f(X))\mathbb{J}^{\mathsf{P}}(X, \mathrm{d}Y).$$

First, note that $\mathcal{A}_{\text{cont}}$ corresponds to the infinitesimal generator of a classical diffusion on the components which are not set to ∞. Hence, we can approximately sample from $(\mathrm{Q}_t)_{t\geq 0}$ by sampling according to the Euler-Maruyama discretization of the associated diffusion, i.e. by setting

$$\mathbf{X}_t \approx \mathbf{X}_0 + tb(\mathbf{X}_0) + \sqrt{t}Z, \tag{7}$$

where Z is a Gaussian random variable.

Similarly, in order to sample from $(\mathrm{J}_t)_{t\geq 0}$, one should sample from the jump process defined as follows. On the interval $[0, \tau)$, we have $\mathbf{X}_t = \mathbf{X}_0$. At time τ, we define $\mathbf{X}_1 \sim \mathbb{J}(\mathbf{X}_0, \cdot)$ and repeat the procedure. In this case τ is defined as an exponential random variable with parameter 1. For $t > 0$ small enough the probability that $t > \tau$ is of order t. Therefore, we sample from J, i.e. the deletion kernel, with probability t. Combining this approximation and (7), we get approximate samplers for $(\mathrm{Q}_t)_{t\geq 0}$ and $(\mathrm{J}_t)_{t\geq 0}$. Under mild assumptions, the Lie-Trotter-Kato formula ensures that for any $t \geq 0$

$$\mathrm{P}_t = \lim_{n\to+\infty} (\mathrm{Q}_{t/n}\mathrm{J}_{t/n})^n.$$

This justifies sampling according to Algorithm 1 (in the case of the forward process).

A.2 Proof of Proposition 1

For the proof of Proposition 1, we first provide a rigorous proof using the notation introduced in A.1. We then follow this with a second proof that aims to be more intuitive using the notation used in the main paper.

A.2.1 Time-reversal for the transdimensional infinitesimal generator and Proof of Proposition 1

We are now going to derive the formula for the time-reversal of the transdimensional infinitesimal generator $\mathcal{A}$, see (5). This corresponds to a rigorous proof of Proposition 1. We refer to Section A.2.2 for a more intuitive, albeit less-rigorous, proof. We start by introducing the kernel $\mathbb{K}^{\mathsf{P}}$ given for any dimension path $\mathsf{D}_0 \to \cdots \to \mathsf{D}_M$, for any $i \in \{0, \dots, M-1\}$, $Y \in \mathsf{D}_{i+1}$ and $\mathsf{A} \in \mathcal{X}$ by

$$\mathbb{K}^{\mathsf{P}}(Y, \mathsf{A}) = \sum_{i=0}^{M-1} \delta_{\mathsf{D}_{i+1}}(\dim(Y)) \int_{\mathsf{A}\cap\mathsf{D}_i} \frac{p_t((X_{\mathsf{D}_i\setminus\mathsf{D}_{i+1}}, Y_{\mathsf{D}_{i+1}})|\dim(\mathbf{X}_t)=\mathsf{D}_i)\mathbb{P}(\dim(\mathbf{X}_t)=\mathsf{D}_i)}{p_t(Y_{\mathsf{D}_{i+1}}|\dim(\mathbf{X}_t)=\mathsf{D}_{i+1})\mathbb{P}(\dim(\mathbf{X}_t)=\mathsf{D}_{i+1})} \mathrm{d}X_{\mathsf{D}_i\setminus\mathsf{D}_{i+1}}.$$

Note that this kernel is the same as the one considered in Proposition 1. It is well-defined under the following assumption.

Assumption 1. *For any $t > 0$ and $\mathsf{D} \subset \{0,1\}^N$, we have that $\mathbf{X}_t$ conditioned to $\dim(\mathbf{X}_t) = \mathsf{D}$ admits a density w.r.t. the $|\mathsf{D}|$-dimensional Lebesgue measure, denoted $p_t(\cdot|\dim(\mathbf{X}_t) = \mathsf{D})$.*

The following result will be key to establish the time-reversal formula.

Lemma 1. *Assume* A1. *Let $\mathsf{A}, \mathsf{B} \in \mathcal{X}$. Let P be a dimension path $\mathsf{D}_0 \to \cdots \to \mathsf{D}_M$ with $M \in \mathbb{N}$. Then, we have*

$$\mathbb{E}[\mathbf{1}_{\mathsf{A}}(\mathbf{X}_t)\mathbb{J}^{\mathsf{P}}(\mathbf{X}_t, \mathsf{B})] = \mathbb{E}[\mathbf{1}_{\mathsf{B}}(\mathbf{X}_t)\mathbb{K}^{\mathsf{P}}(\mathbf{X}_t, \mathsf{A})].$$

Proof. Let $\mathsf{A}, \mathsf{B} \in \mathcal{X}$. We have

$$\begin{aligned}
&\mathbb{E}[\mathbf{1}_{\mathsf{A}}(\mathbf{X}_t)\mathbb{J}^{\mathsf{P}}(\mathbf{X}_t, \mathsf{B})] = \textstyle\sum_{i=0}^{M-1} \mathbb{E}[\mathbf{1}_{\mathsf{A}}(\mathbf{X}_t)\delta_{\mathsf{D}_i}(\dim(\mathbf{X}_t))\mathbf{1}_{\mathsf{B}}((\mathbf{X}_t)_{\mathsf{D}_{i+1}})] \\
&\quad = \textstyle\sum_{i=0}^{M-1} \int_{\mathsf{A}\cap\mathsf{D}_i} p_t(X_{\mathsf{D}_i}|\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathbb{P}(\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathbf{1}_{\mathsf{B}}(X_{\mathsf{D}_{i+1}})\mathrm{d}X_{\mathsf{D}_i} \\
&\quad = \textstyle\sum_{i=0}^{M-1} \int_{\mathsf{A}\cap\mathsf{D}_i} p_t(X_{\mathsf{D}_i}|\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathbb{P}(\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathbf{1}_{\mathsf{B}}(X_{\mathsf{D}_{i+1}})\mathrm{d}X_{\mathsf{D}_{i+1}}\mathrm{d}X_{\mathsf{D}_i\setminus\mathsf{D}_{i+1}} \\
&\quad = \textstyle\sum_{i=0}^{M-1} \int_{\mathsf{B}\cap\mathsf{D}_{i+1}} \mathbf{1}_{\mathsf{B}}(X_{\mathsf{D}_{i+1}}) \\
&\qquad\quad \times \big(\int_{\mathsf{A}\cap\mathsf{D}_i} \mathbf{1}_{\mathsf{A}}(X_{\mathsf{D}_i}) p_t(X_{\mathsf{D}_i}|\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathbb{P}(\dim(\mathbf{X}_t) = \mathsf{D}_i)\mathrm{d}X_{\mathsf{D}_i\setminus\mathsf{D}_{i+1}}\big)\mathrm{d}X_{\mathsf{D}_{i+1}} \\
&\quad = \textstyle\sum_{i=0}^{M-1} \int_{\mathsf{B}\cap\mathsf{D}_{i+1}} \mathbf{1}_{\mathsf{B}}(X_{\mathsf{D}_{i+1}}) \\
&\qquad\quad \times \mathbb{K}^{\mathsf{P}}(X_{\mathsf{D}_{i+1}}, \mathsf{A}) p_t(X_{\mathsf{D}_{i+1}}|\dim(\mathbf{X}_t) = \mathsf{D}_{i+1})\mathbb{P}(\dim(\mathbf{X}_t) = \mathsf{D}_{i+1})\mathrm{d}X_{\mathsf{D}_{i+1}} \\
&\quad = \textstyle\sum_{i=0}^{M-1} \mathbb{E}[\delta_{\mathsf{D}_{i+1}}(\dim(\mathbf{X}_t))\mathbb{K}^{\mathsf{P}}(\mathbf{X}_t, \mathsf{A})\mathbf{1}_{\mathsf{B}}(\mathbf{X}_t)],
\end{aligned}$$

which concludes the proof. □

Lemma 1 shows that $\mathbb{K}^{\mathsf{P}}$ verifies the *flux equation* associated with $\mathbb{J}^{\mathsf{P}}$. The flux equation is the discrete state-space equivalent of the classical time-reversal formula for continuous state-space. We refer to [45] for a rigorous treatment of time-reversal with jumps under entropic conditions.

We are also going to consider the following assumption which ensures that the integration by part formula is valid.

Assumption 2. *For any $t > 0$ and $i \in \{1, \dots, N\}$, $\mathbf{X}_t$ admits a smooth density w.r.t. the Nd-dimensional Lebesgue measure denoted p_t and we have that for any $f, h \in \mathrm{C}_b^2((\mathbb{R}^d)^N)$ for any $u \in [0, t]$ and $i \in \{1, \dots, N\}$*

$$\begin{aligned}\mathbb{E}[\delta_{\mathbb{R}^d}((\mathbf{X}_u)_i)\langle \nabla_{x_i} f(\mathbf{X}_u), \nabla_{x_i} h(\mathbf{X}_u)\rangle] \\ = -\mathbb{E}[\delta_{\mathbb{R}^d}((\mathbf{X}_u)_i) h(\mathbf{X}_u)(\Delta_{x_i} f(\mathbf{X}_u) + \langle \nabla_{x_i} \log p_u(\mathbf{X}_u), \nabla_{x_i} f(\mathbf{X}_u)\rangle)].\end{aligned}$$

The second assumption ensures that we can apply the backward Kolmogorov evolution equation.

Assumption 3. *For any $g \in \mathrm{C}^2(\mathsf{X})$ and $t > 0$, we have that for any $u \in [0, t]$ and $X \in \mathsf{X}$, $\partial_u g(u, X) + \mathcal{A}(g)(u, X) = 0$, where for any $u \in [0, t]$ and $X \in \mathsf{X}$, $g(u, X) = \mathbb{E}[g(\mathbf{X}_t) \mid \mathbf{X}_u = X]$.*

We refer to [19] for conditions under A2 and A3 are valid in the setting of diffusion processes.

Proposition 4. *Assume* A1*,* A2 *and* A3*. Assume that there exists a Markov process $(\mathbf{X}_t)_{t\geq 0}$ solution of the martingale problem associated with* (6)*. Let $T > 0$ and consider $(\mathbf{Y}_t)_{t\in[0,T]} = (\mathbf{X}_{T-t})_{t\in[0,T]}$. Then $(\mathbf{Y}_t)_{t\in[0,T]}$ is solution to the martingale problem associated with $\mathcal{R}$, where for any $f \in \mathrm{C}^2(\mathsf{X})$, $t \in (0, T)$ and $x \in \mathsf{X}$ we have*

$$\begin{aligned}\mathcal{R}(f)(t, X) = \textstyle\sum_{i=1}^N \{-\langle b(X_i) + \nabla_{x_i} \log p_t(X), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)\}\delta_{\mathbb{R}^d}(X_i) \\ + \textstyle\int_{\mathsf{X}} (f(Y) - f(X))\mathbb{K}(X, \mathrm{d}Y).\end{aligned}$$

Proof. Let $f, g \in \mathrm{C}^2(\mathsf{X})$. In what follows, we show that for any $s, t \in [0, T]$ with $t \geq s$

$$\mathbb{E}[(f(\mathbf{Y}_t) - f(\mathbf{Y}_s))g(\mathbf{Y}_s)] = \mathbb{E}[g(\mathbf{Y}_s) \textstyle\int_s^t \mathcal{R}(f)(u, \mathbf{Y}_u)\mathrm{d}u].$$

More precisely, we show that for any $s, t \in [0, T]$ with $t \geq s$

$$\mathbb{E}[(f(\mathbf{X}_t) - f(\mathbf{X}_s))g(\mathbf{X}_t)] = \mathbb{E}[-g(\mathbf{X}_t) \textstyle\int_s^t \mathcal{R}(f)(u, \mathbf{X}_u)\mathrm{d}u].$$

Let $s, t \in [0, T]$, with $t \geq s$. Next, we denote for any $u \in [0, t]$ and $X \in \mathsf{X}$, $g(u, X) = \mathbb{E}[g(\mathbf{X}_t) \mid \mathbf{X}_u = X]$. Using A3, we have that for any $u \in [0, t]$ and $X \in \mathsf{X}$, $\partial_u g(u, X) + \mathcal{A}(g)(u, X) = 0$, i.e. g satisfies the backward Kolmogorov equation. For any $u \in [0, t]$ and $X \in \mathsf{X}$, we have

$$\begin{aligned}\mathcal{A}(fg)(u, X) &= \partial_u g(u, X) f(X) + \textstyle\sum_{i=1}^N (\langle b(X_i), \nabla_{x_i} g(u, X)\rangle + \tfrac{1}{2}\Delta_{x_i} g(u, X_i)) f(X)\delta_{\mathbb{R}^d}(X_i) \\ &\quad + \textstyle\sum_{i=1}^N (\langle b(X_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)) g(u, X)\delta_{\mathbb{R}^d}(X_i) \\ &\quad + \textstyle\sum_{i=1}^N \delta_{\mathbb{R}^d}(X_i)\langle \nabla_{x_i} f(X), \nabla_{x_i} g(u, X)\rangle + \mathbb{J}(X, fg) \\ &= \partial_u g(u, X) f(X) + \mathcal{A}(g)(u, X) f(X) + \mathbb{J}(X, fg) - \mathbb{J}(X, g) f(X) \\ &\quad + \textstyle\sum_{i=1}^N (\langle b(X_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)) g(u, X)\delta_{\mathbb{R}^d}(X_i) \\ &\quad + \textstyle\sum_{i=1}^N \delta_{\mathbb{R}^d}(X_i)\langle \nabla_{x_i} f(X), \nabla_{x_i} g(u, X)\rangle \\ &= \textstyle\sum_{i=1}^N (\langle b(X_i), \nabla_{x_i} f(X)\rangle + \tfrac{1}{2}\Delta_{x_i} f(X)) g(u, X)\delta_{\mathbb{R}^d}(X_i) \\ &\quad + \textstyle\sum_{i=1}^N \delta_{\mathbb{R}^d}(X_i)\langle \nabla_{x_i} f(X), \nabla_{x_i} g(u, X)\rangle + \mathbb{J}(X, fg) - \mathbb{J}(X, g) f(X). \qquad (8)\end{aligned}$$

Using A2, we have that for any $u \in [0, t]$ and $i \in \{1, \dots, N\}$

$$\begin{aligned}\mathbb{E}[\delta_{\mathbb{R}^d}((\mathbf{X}_u)_i)\langle \nabla_{x_i} f(\mathbf{X}_u), \nabla_{x_i} g(u, \mathbf{X}_u)\rangle] \\ = -\mathbb{E}[\delta_{\mathbb{R}^d}((\mathbf{X}_u)_i) g(u, \mathbf{X}_u)(\Delta_{x_i} f(\mathbf{X}_u) + \langle \nabla_{x_i} \log p_u(\mathbf{X}_u), \nabla_{x_i} f(\mathbf{X}_u)\rangle)]. \qquad (9)\end{aligned}$$

In addition, we have that for any $X \in \mathsf{X}$ and $u \in [0, t]$, $\mathbb{J}(X, fg) - \mathbb{J}(X, g) f(X) = \int_{\mathsf{X}} g(u, Y)(f(Y) - f(X))\mathbb{J}(X, \mathrm{d}Y)$. Using Lemma 1, we get

$$\mathbb{E}[\mathbb{J}(\mathbf{X}_u, fg) - \mathbb{J}(\mathbf{X}_u, f) g(u, \mathbf{X}_u)] = -\mathbb{E}[g(u, \mathbf{X}_u)\mathbb{K}(\mathbf{X}_u, f)]. \qquad (10)$$

Therefore, using (8), (9) and (10), we have

$$\mathbb{E}[\mathcal{A}(fg)(u, \mathbf{X}_u)] = \mathbb{E}[-\mathcal{R}(f)(u, \mathbf{X}_u)g(u, \mathbf{X}_u)].$$

Finally, we have

$$\begin{aligned}\mathbb{E}[(f(\mathbf{X}_t) - f(\mathbf{X}_s))g(\mathbf{X}_t)] &= \mathbb{E}[g(t, \mathbf{X}_t)f(\mathbf{X}_t) - f(\mathbf{X}_s)g(s, \mathbf{X}_s)] \\ &= \mathbb{E}[\textstyle\int_s^t \mathcal{A}(fg)(u, \mathbf{X}_u)\mathrm{d}u] \\ &= -\mathbb{E}[\textstyle\int_s^t \mathcal{R}(f)(u, \mathbf{X}_u)g(u, \mathbf{X}_u)\mathrm{d}u] = -\mathbb{E}[g(\mathbf{X}_t)\textstyle\int_s^t \mathcal{R}(f)(u, \mathbf{X}_u)\mathrm{d}u],\end{aligned}$$

which concludes the proof. □

A.2.2 Intuitive Proof of Proposition 1

We recall Proposition 1.

Proposition 1. *The time reversal of a forward jump diffusion process given by drift $\overrightarrow{\mathbf{b}}_t$, diffusion coefficient $\overrightarrow{g}_t$, rate $\overrightarrow{\lambda}_t(n)$ and transition kernel $\sum_{i=1}^n K^{\text{del}}(i|n)\delta_{\text{del}(\mathbf{X},i)}(\mathbf{Y})$ is given by a jump diffusion process with drift $\overleftarrow{\mathbf{b}}_t^*(\mathbf{X})$, diffusion coefficient $\overleftarrow{g}_t^*$, rate $\overleftarrow{\lambda}_t^*(\mathbf{X})$ and transition kernel $\int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X})\delta_{\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)}(\mathbf{Y})\mathrm{d}\mathbf{y}^{\text{add}}$ as defined below*

$$\begin{aligned}&\overleftarrow{\mathbf{b}}_t^*(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X}) - \nabla_{\mathbf{x}} \log p_t(\mathbf{X}), \quad \overleftarrow{g}_t^* = \overrightarrow{g}_t, \\ &\overleftarrow{\lambda}_t^*(\mathbf{X}) = \overrightarrow{\lambda}_t(n+1)\frac{\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i))\mathrm{d}\mathbf{y}^{\text{add}}}{p_t(\mathbf{X})}, \\ &A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X}) \propto p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i))K^{\text{del}}(i|n+1).\end{aligned}$$

Diffusion part. Using standard diffusion models arguments such as [18] [45], we get

$$\overleftarrow{\mathbf{b}}_t^*(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X}) - \nabla_{\mathbf{x}} \log p_t(\mathbf{X}|n).$$

Jump part. We use the flux equation from [45] which intuitively relates the probability flow going in the forward direction with the probability flow going the backward direction with equality being achieved at the time reversal.

$$\begin{aligned}&p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X})\overleftarrow{K}_t^*(\mathbf{Y}|\mathbf{X}) = p_t(\mathbf{Y})\overrightarrow{\lambda}_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{X}|\mathbf{Y}) \\ &p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X})\textstyle\int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X})\delta_{\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)}(\mathbf{Y})\mathrm{d}\mathbf{y}^{\text{add}} \\ &\quad = p_t(\mathbf{Y})\overrightarrow{\lambda}_t(\mathbf{Y})\textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\delta_{\text{del}(\mathbf{Y},i)}(\mathbf{X}). \qquad (11)\end{aligned}$$

To find $\overleftarrow{\lambda}_t^*(\mathbf{X})$, we sum and integrate both sides over m and $\mathbf{y}$, with $\mathbf{Y} = (m, \mathbf{y})$,

$$\begin{aligned}&\textstyle\sum_{m=1}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X})\int_{\mathbf{y}^{\text{add}}} \sum_{i=1}^{n+1} A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X})\delta_{\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)}(\mathbf{Y})\mathrm{d}\mathbf{y}^{\text{add}}\mathrm{d}\mathbf{y} \\ &= \textstyle\sum_{m=1}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} p_t(\mathbf{Y})\overrightarrow{\lambda}_t(\mathbf{Y})\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\delta_{\text{del}(\mathbf{Y},i)}(\mathbf{X})\mathrm{d}\mathbf{y}.\end{aligned}$$

Now we use the fact that $\delta_{\text{del}(\mathbf{Y},i)}(\mathbf{X})$ is 0 for any $m \neq n+1$,

$$\begin{aligned}p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X}) &= \overrightarrow{\lambda}_t(n+1)\textstyle\int_{\mathbf{y}\in\mathbb{R}^{(n+1)d}} p_t(\mathbf{Y})\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\delta_{\text{del}(\mathbf{Y},i)}(\mathbf{X})\mathrm{d}\mathbf{y} \\ &= \overrightarrow{\lambda}_t(n+1)\textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}\in\mathbb{R}^{(n+1)d}} p_t(\mathbf{Y})\delta_{\text{del}(\mathbf{Y},i)}(\mathbf{X})\mathrm{d}\mathbf{y}.\end{aligned}$$

Now letting $\mathbf{Y} = \text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)$,

$$\begin{aligned}p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X}) &= \overrightarrow{\lambda}_t(n+1)\textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i))\mathrm{d}\mathbf{y}^{\text{add}} \\ \overleftarrow{\lambda}_t^*(\mathbf{X})x &= \overrightarrow{\lambda}_t(n+1)\tfrac{\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i))\mathrm{d}\mathbf{y}^{\text{add}}}{p_t(\mathbf{X})}.\end{aligned}$$

To find $A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X})$, we start from (11) and set $\mathbf{Y} = \text{ins}(\mathbf{X}, \mathbf{z}^{\text{add}}, j)$ to get

$$p_t(\mathbf{X})\overleftarrow{\lambda}_t^*(\mathbf{X})A_t^*(\mathbf{z}^{\text{add}}, j|\mathbf{X}) = p_t(\mathbf{Y})\overrightarrow{\lambda}_t(n+1)K^{\text{del}}(j|n+1).$$

By inspection, we see immediately that

$$A_t^*(\mathbf{z}^{\text{add}}, j|\mathbf{X}) \propto p_t(\text{ins}(\mathbf{X}, \mathbf{z}^{\text{add}}, j)K^{\text{del}}(j|n+1).$$

With a re-labeling of $\mathbf{z}^{\text{add}}$ and j we achieve the desired form

$$A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X}) \propto p_t(\text{ins}(\mathbf{X}, \mathbf{y}^{\text{add}}, i))K^{\text{del}}(i|n+1).$$

A.3 Proof of Proposition 2

In this section we prove Proposition 2 using the notation from the main paper by following the framework of [17]. We operate on a state space $\mathcal{X} = \bigcup_{n=1}^{N}\{n\} \times \mathbb{R}^{nd}$. On this space the gradient operator $\nabla : \mathcal{C}(\mathcal{X}, \mathbb{R}) \to \mathcal{C}(\mathcal{X}, \mathcal{X})$ is defined as $\nabla f(\mathbf{X}) = \nabla_{\mathbf{x}}^{(nd)} f(\mathbf{X})$ where $\nabla_{\mathbf{x}}^{(nd)}$ is the standard gradient operator defined as $\mathcal{C}(\mathbb{R}^{nd}, \mathbb{R}) \to \mathcal{C}(\mathbb{R}^{nd}, \mathbb{R}^{nd})$ with respect to $\mathbf{x} \in \mathbb{R}^{nd}$. We will write integration with respect to a probability measure defined on $\mathcal{X}$ as an explicit sum over the number of components and integral over $\mathbb{R}^{nd}$ with respect to a probability density defined on $\mathbb{R}^{nd}$ i.e. $\int_{\mathbf{X}} f(\mathbf{X})\mu(\mathrm{d}\mathbf{X}) = \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})p(n)p(\mathbf{x}|n)\mathrm{d}\mathbf{x}$ where, for $A \subset \mathbb{R}^{nd}$, $\int_{(n,A)}\mu(d\mathbf{X}) = \int_{\mathbf{x}\in A} p(n)p(\mathbf{x}|n)\mathrm{d}\mathbf{x}$. We will write $p(\mathbf{X})$ as shorthand for $p(n)p(\mathbf{x}|n)$.

Following, [17], we start by augmenting our space with a time variable so that operators become time inhomegeneous on the extended space. We write this as $\bar{\mathbf{X}} = (\mathbf{X}, t)$ where $\bar{\mathbf{X}}$ lives in the extended space $\mathcal{S} = \mathcal{X} \times \mathbb{R}_{\geq 0}$. In the proof, we use the infinitesimal generators for the the forward and backward processes. An infinitesimal generator is defined as

$$\mathcal{A}(f)(\bar{\mathbf{X}}) = \lim_{t\to 0}\frac{\mathbb{E}_{p_{t|0}(\bar{\mathbf{Y}}|\bar{\mathbf{X}})}[f(\bar{\mathbf{Y}})] - f(\bar{\mathbf{X}})}{t}$$

and can be understood as a probabilistic version of a derivative. For our process on the augmented space $\mathcal{S}$, our generators decompose as $\mathcal{A} = \partial_t + \hat{\mathcal{A}}_t$ where $\hat{\mathcal{A}}_t$ operates only on the spatial components of $\bar{\mathbf{X}}$ i.e. $\mathbf{X}$ [17].

We now define the spatial infinitesimal generators for our forward and backward process. We will change our treatment of the time variable compared to the main text. Both our forward and backward processes will run from $t = 0$ to $t = T$, with the true time reversal of $\mathbf{X}$ following the forward process satisfying $(\mathbf{Y}_t)_{t\in[0,T]} = (\mathbf{X}_{T-t})_{t\in[0,T]}$. Further, we will write $\overrightarrow{g}_t$ as g_t and $\overleftarrow{g}_t = g_{T-t}$ as we do not learn g and this is the optimal relation from the time reversal. We define

$$\hat{\mathcal{L}}_t(f)(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X})\cdot\nabla f(\mathbf{X}) + \tfrac{1}{2}g_t^2\Delta f(\mathbf{X}) + \overrightarrow{\lambda}_t(\mathbf{X})\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y},$$

as well as

$$\hat{\mathcal{K}}_t(f)(\mathbf{X}) = \overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X})\cdot\nabla f(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2\Delta f(\mathbf{X}) + \overleftarrow{\lambda}_t^{\theta}(\mathbf{X})\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}$$

where $\Delta = (\nabla\cdot\nabla)$ is the Laplace operator and δ is a dirac delta on $\mathcal{X}$ i.e. $\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\delta_{\mathbf{X}}(\mathbf{Y})\mathrm{d}\mathbf{y} = 1$ and $\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})\delta_{\mathbf{X}}(\mathbf{Y})\mathrm{d}\mathbf{y} = f(\mathbf{X})$.

Verifying Assumption 1. The first step in the proof is to verify Assumption 1 in [17]. Letting $\nu_t(\mathbf{X}) = p_{T-t}(\mathbf{X})$, we assume we can write $\partial_t p_t(\mathbf{X}) = \hat{\mathcal{K}}_t^* p_t(\mathbf{X})$ in the form $\mathcal{M}\nu + c\nu = 0$ for some function $c : \mathcal{S} \to \mathbb{R}$, where $\mathcal{M}$ is the generator of another auxiliary process on $\mathcal{S}$ and $\hat{\mathcal{K}}_t^*$ is the adjoint operator which satisfies $\langle\hat{\mathcal{K}}_t^* f, h\rangle = \langle f, \hat{\mathcal{K}}_t h\rangle$ i.e.

$$\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\hat{\mathcal{K}}_t^*(f)(\mathbf{X})\mathrm{d}\mathbf{x} = \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\hat{\mathcal{K}}_t(h)(\mathbf{X})\mathrm{d}\mathbf{x}$$

We now find $\hat{\mathcal{K}}_t^*$. We start by substituting in the form for $\hat{\mathcal{K}}_t$,

$$\begin{aligned}\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\hat{\mathcal{K}}_t(h)(\mathbf{X})\mathrm{d}\mathbf{x} = \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\{(\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{x})\cdot\nabla h)(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2\Delta h(\mathbf{X}) + \\ \overleftarrow{\lambda}_t^{\theta}(\mathbf{X})\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} h(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}\}\mathrm{d}\mathbf{x}\end{aligned}$$

We first focus on the RHS terms corresponding to the diffusion part of the process

$$\begin{aligned}&\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\{(\overleftarrow{\mathbf{b}}_t^{\theta}\cdot\nabla h)(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2\Delta h(\mathbf{X})\}\mathrm{d}\mathbf{x}\\ &= \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})(\overleftarrow{\mathbf{b}}_t^{\theta}\cdot\nabla h)(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2 f(\mathbf{X})\nabla\cdot\nabla h(\mathbf{X})\mathrm{d}\mathbf{x}\\ &= \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})(\overleftarrow{\mathbf{b}}_t^{\theta}\cdot\nabla h)(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2 h(\mathbf{X})\nabla\cdot\nabla f(\mathbf{X})\mathrm{d}\mathbf{x}\\ &= \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} -h(\mathbf{X})\nabla\cdot(f\overleftarrow{\mathbf{b}}_t^{\theta})(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2 h(\mathbf{X})\nabla\cdot\nabla f(\mathbf{X})\mathrm{d}\mathbf{x}\\ &= \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\{-\nabla\cdot(f\overleftarrow{\mathbf{b}}_t^{\theta})(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2\nabla\cdot\nabla f(\mathbf{X})\}\mathrm{d}\mathbf{x}\\ &= \sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\{-f(\mathbf{X})\nabla\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X}) - \nabla f(\mathbf{X})\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X}) + \tfrac{1}{2}g_{T-t}^2\nabla\cdot\nabla f(\mathbf{X})\}\mathrm{d}\mathbf{x}.\end{aligned}$$

where we apply integration by parts twice to arrive at the third line and once to arrive at the fourth line. We now focus on the RHS term corresponding to the jump part of the process

$$\begin{aligned}
&\textstyle\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\{\overleftarrow{\lambda}_t^{\theta}(\mathbf{X})\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} h(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{Y}|\mathbf{X})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}\}\mathrm{d}\mathbf{x}\\
&=\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} h(\mathbf{Y})\{\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} f(\mathbf{X})\overleftarrow{\lambda}_t^{\theta}(\mathbf{X})(\overleftarrow{K}_t^{\theta}(\mathbf{Y}|\mathbf{X})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{x}\}\mathrm{d}\mathbf{y}\\
&=\textstyle\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\{\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})\overleftarrow{\lambda}_t^{\theta}(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{X}|\mathbf{Y})-\delta_{\mathbf{Y}}(\mathbf{X}))\mathrm{d}\mathbf{y}\}\mathrm{d}\mathbf{x},
\end{aligned}$$

where on the last line we have relabelled $\mathbf{X}$ to $\mathbf{Y}$ and $\mathbf{Y}$ to $\mathbf{X}$. Putting both re-arranged forms for the RHS together, we obtain

$$\begin{aligned}
\textstyle\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\hat{\mathcal{K}}_t^*(f)(\mathbf{X})\mathrm{d}\mathbf{x} =&\\
&\textstyle\sum_{n=1}^{N}\int_{\mathbf{x}\in\mathbb{R}^{nd}} h(\mathbf{X})\{-f(\mathbf{X})\nabla\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X})-\nabla f(\mathbf{X})\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X})+\tfrac{1}{2}g_{T-t}^2\nabla\cdot\nabla f(\mathbf{X})+\\
&\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})\overleftarrow{\lambda}_t^{\theta}(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{X}|\mathbf{Y})-\delta_{\mathbf{Y}}(\mathbf{X}))\mathrm{d}\mathbf{y}\}\mathrm{d}\mathbf{x}.
\end{aligned}$$

We therefore have

$$\begin{aligned}
\hat{\mathcal{K}}_t^*(f)(\mathbf{X}) =& -f(\mathbf{X})\nabla\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X})-\nabla f(\mathbf{X})\cdot\overleftarrow{\mathbf{b}}_t^{\theta}(\mathbf{X})+\tfrac{1}{2}g_{T-t}^2\Delta f(\mathbf{X})+\\
&\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})\overleftarrow{\lambda}_t^{\theta}(\mathbf{Y})(\overleftarrow{K}_t^{\theta}(\mathbf{X}|\mathbf{Y})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}.
\end{aligned}$$

Now we re-write $\partial_t p_t(\mathbf{X})=\hat{\mathcal{K}}_t^* p_t(\mathbf{x})$ in the form $\mathcal{M}\nu+c\nu=0$. We start by re-arranging

$$\partial_t p_t(\mathbf{X})=\hat{\mathcal{K}}_t^* p_t(\mathbf{X}) \implies 0=\partial_t\nu_t(\mathbf{X})+\hat{\mathcal{K}}_{T-t}^*\nu_t(\mathbf{X}).$$

Substituting in our form for $\hat{\mathcal{K}}_t^*$ we obtain

$$\begin{aligned}
0 =&\partial_t\nu_t(\mathbf{X})-\nu_t(\mathbf{X})\nabla\cdot\overleftarrow{\mathbf{b}}_{T-t}^{\theta}(\mathbf{X})-\overleftarrow{\mathbf{b}}_{T-t}^{\theta}(\mathbf{X})\cdot\nabla\nu_t(\mathbf{X})+\tfrac{1}{2}g_t^2\Delta\nu_t(\mathbf{X})\\
&+\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\nu_t(\mathbf{Y})\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{Y})(\overleftarrow{K}_{T-t}^{\theta}(\mathbf{X}|\mathbf{Y})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}
\end{aligned}\tag{12}$$

We define our auxiliary process to have generator $\mathcal{M}=\partial_t+\hat{\mathcal{M}}_t$ with

$$\hat{\mathcal{M}}_t(f)(\mathbf{X})=\mathbf{b}_t^M(\mathbf{X})\cdot\nabla f(\mathbf{X})+\tfrac{1}{2}g_t^2\Delta f(\mathbf{X})+\lambda_t^M(\mathbf{X})\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}} f(\mathbf{Y})(K_t^M(\mathbf{Y}|\mathbf{X})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}$$

which is a jump diffusion process with drift $\mathbf{b}_t^M$, diffusion coefficient g_t, rate λ_t^M and transition kernel K_t^M. Then if we have $\mathbf{b}_t^M=-\overleftarrow{\mathbf{b}}_{T-t}^{\theta}$,

$$\lambda_t^M(\mathbf{X})=\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{Y})\overleftarrow{K}_{T-t}^{\theta}(\mathbf{X}|\mathbf{Y})\mathrm{d}\mathbf{y},\tag{13}$$

and

$$K_t^M(\mathbf{Y}|\mathbf{X})\propto\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{Y})\overleftarrow{K}_{T-t}^{\theta}(\mathbf{X}|\mathbf{Y}).\tag{14}$$

Then we have (12) can be rewritten as

$$\begin{aligned}
0 =&\partial_t\nu_t(\mathbf{X})-\nu_t(\mathbf{X})\nabla\cdot\overleftarrow{\mathbf{b}}_{T-t}^{\theta}(\mathbf{X})+\mathbf{b}_t^M(\mathbf{X})\cdot\nabla\nu_t(\mathbf{X})+\tfrac{1}{2}g_t^2\Delta\nu_t(\mathbf{X})\\
&-\nu_t(\mathbf{X})\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{X})+\nu_t(\mathbf{X})\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{Y})\overleftarrow{K}_{T-t}^{\theta}(\mathbf{X}|\mathbf{Y})\mathrm{d}\mathbf{y}+\\
&\lambda_t^M(\mathbf{x})\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\nu_t(\mathbf{Y})(K_t^M(\mathbf{Y}|\mathbf{X})-\delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}
\end{aligned}$$

which is in the form $\mathcal{M}(\nu)(\bar{\mathbf{X}})+c(\bar{\mathbf{X}})\nu(\bar{\mathbf{X}})=0$ if we let

$$c(\bar{\mathbf{X}})=-\nabla\cdot\overleftarrow{\mathbf{b}}_{T-t}^{\theta}(\mathbf{X})-\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{X})+\textstyle\sum_{m=1}^{N}\int_{\mathbf{y}\in\mathbb{R}^{md}}\overleftarrow{\lambda}_{T-t}^{\theta}(\mathbf{Y})\overleftarrow{K}_{T-t}^{\theta}(\mathbf{X}|\mathbf{Y})\mathrm{d}\mathbf{y}.$$

Verifying Assumption 2. Now that we have verified Assumption 1, the second step in the proof is Assumption 2 from [17]. We assume there is a bounded measurable function $\alpha:\mathcal{S}\to(0,\infty)$ such

that $\alpha\mathcal{M}f = \mathcal{L}(f\alpha) - f\mathcal{L}\alpha$ for all functions $f : \mathcal{X} \to \mathbb{R}$ such that $f \in \mathcal{D}(\mathcal{M})$ and $f\alpha \in \mathcal{D}(\mathcal{L})$. Substituting in $\mathcal{M}$ and $\mathcal{L}$ we get

$$\begin{aligned}
&\alpha_t(\mathbf{X})[\partial_t f(\mathbf{X}) - \overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X}) \cdot \nabla f(\mathbf{X}) \\
&\quad + \tfrac{1}{2} g_t^2 \Delta f(\mathbf{X}) + \lambda_t^M(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})(K_t^M(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}] \\
&= \partial_t(f\alpha_t)(\mathbf{X}) + \overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla(f\alpha_t)(\mathbf{X}) + \tfrac{1}{2} g_t^2 \Delta(f\alpha_t)(\mathbf{X}) \\
&\quad + \overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\alpha_t(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y} \\
&\quad - f(\mathbf{X})[\partial_t \alpha_t(\mathbf{X}) + \overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla \alpha_t(\mathbf{X}) + \tfrac{1}{2} g_t^2 \Delta \alpha_t(\mathbf{X}) \\
&\quad + \overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} \alpha_t(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}]
\end{aligned} \tag{15}$$

Since f does not depend on time, $\partial_t f(\mathbf{X}) = 0$ and $\partial_t(f\alpha_t)(\mathbf{X}) = f(\mathbf{X})\partial_t\alpha_t(\mathbf{X})$ thus the ∂_t terms on the RHS also cancel out. Comparing terms on the LHS and RHS relating to the diffusion part of the process we obtain

$$\begin{aligned}
&-\alpha_t(\mathbf{X})(\overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X}) \cdot \nabla f(\mathbf{X})) + \tfrac{1}{2}\alpha_t(\mathbf{X}) g_t^2 \Delta f(\mathbf{X}) = \\
&\overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla(f\alpha_t)(\mathbf{X}) + \tfrac{1}{2} g_t^2 \Delta(f\alpha_t)(\mathbf{X}) - f(\mathbf{X})\overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla \alpha_t(\mathbf{X}) - \tfrac{1}{2} f(\mathbf{X}) g_t^2 \Delta \alpha_t(\mathbf{X}).
\end{aligned}$$

Therefore, we get

$$\begin{aligned}
&-\alpha_t(\mathbf{X})(\overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X}) \cdot \nabla f(\mathbf{X})) + \tfrac{1}{2}\alpha_t(\mathbf{X}) g_t^2 \Delta f(\mathbf{X}) = \\
&\quad \overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot (f(\mathbf{X})\nabla \alpha_t(\mathbf{X}) + \alpha_t(\mathbf{X})\nabla f(\mathbf{X})) \\
&\quad + \tfrac{1}{2} g_t^2 \big(2\nabla f(\mathbf{X}) \cdot \nabla \alpha_t(\mathbf{X}) + f(\mathbf{X})\Delta \alpha_t(\mathbf{X}) + \alpha_t(\mathbf{X})\Delta f(\mathbf{X})\big) \\
&\quad - f(\mathbf{X})\overrightarrow{\mathbf{b}}_t((\mathbf{X}) \cdot \nabla \alpha_t(\mathbf{X}) - \tfrac{1}{2} f(\mathbf{X}) g_t^2 \Delta \alpha_t(\mathbf{X}).
\end{aligned}$$

Simplifying the above expression, we get

$$\begin{aligned}
-\alpha_t(\mathbf{X})(\overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X}) \cdot \nabla f(\mathbf{X})) &= \alpha_t(\mathbf{X})\overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla f(\mathbf{X}) + g_t^2 \nabla f(\mathbf{X}) \cdot \nabla \alpha_t(\mathbf{X}) \\
(-\alpha_t(\mathbf{X})\overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X})) \cdot \nabla f(\mathbf{X}) &= (\alpha_t(\mathbf{X})\overrightarrow{\mathbf{b}}_t(\mathbf{X}) + g_t^2 \nabla \alpha_t(\mathbf{X})) \cdot \nabla f(\mathbf{X}).
\end{aligned}$$

This is true for any f implying

$$-\alpha_t(\mathbf{X})\overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X}) = \alpha_t(\mathbf{X})\overrightarrow{\mathbf{b}}_t(\mathbf{X}) + g_t^2 \nabla \alpha_t(\mathbf{X}).$$

This implies that $\alpha_t(\mathbf{X})$ satisfies

$$\nabla \log \alpha_t(\mathbf{X}) = -\tfrac{1}{g_t^2}(\overrightarrow{\mathbf{b}}_t(\mathbf{X}) + \overleftarrow{\mathbf{b}}^{\theta}_{T-t}(\mathbf{X})) \tag{16}$$

Comparing terms from the LHS and RHS of (15) relating to the jump part of the process we obtain

$$\begin{aligned}
&\alpha_t(\mathbf{X})\lambda_t^M(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})(K_t^M(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y} = \\
&\quad \overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\alpha_t(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y} \\
&\quad - f(\mathbf{X})\overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} \alpha_t(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}.
\end{aligned}$$

Hence, we have

$$\begin{aligned}
&\alpha_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\lambda_t^M(\mathbf{X})K_t^M(\mathbf{Y}|\mathbf{X})\mathrm{d}\mathbf{y} - \alpha_t(\mathbf{X})\lambda_t^M(\mathbf{X})f(\mathbf{X}) = \\
&\quad \overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\alpha_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})\mathrm{d}\mathbf{y} \\
&\quad - f(\mathbf{X})\overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} \alpha_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})\mathrm{d}\mathbf{y}.
\end{aligned}$$

Recalling the definitions of $\lambda_t^M(\mathbf{X})$ and $K^M(\mathbf{Y}|\mathbf{X})$, (13) and (14), we get

$$\begin{aligned}
&\alpha_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y})\overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}|\mathbf{Y})\mathrm{d}\mathbf{y} \\
&\qquad - \alpha_t(\mathbf{X})f(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y})\overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}|\mathbf{Y})\mathrm{d}\mathbf{y} = \\
&\quad \overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})\alpha_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})\mathrm{d}\mathbf{y} \\
&\quad - f(\mathbf{X})\overrightarrow{\lambda}_t(\mathbf{X}) \textstyle\sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} \alpha_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})\mathrm{d}\mathbf{y}.
\end{aligned}$$

This equality is satisfied if $\alpha_t(\mathbf{X})$ follows the following relation

$$\alpha_t(\mathbf{Y}) = \alpha_t(\mathbf{X}) \frac{\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) \overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}|\mathbf{Y})}{\overrightarrow{\lambda}_t(\mathbf{X}) \overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})} \quad \text{for } n \neq m \tag{17}$$

We only require this relation to be satisfied for $n \neq m$ because both $\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X})$ and $\overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}|\mathbf{Y})$ are 0 for $n = m$. We note at this point, as in [17], that if we have $\alpha_t(\mathbf{X}) = 1/p_t(\mathbf{X})$ and $\overleftarrow{\lambda}^{\theta}_{T-t}$ and $\overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}|\mathbf{Y})$ equal to the true time-reversals, then both (16), and (17) are satisfied. However, $\alpha_t(\mathbf{X}) = 1/p_t(\mathbf{X})$ is not the only α_t to satisfy these equations. (16) and (17) can be thought of as enforcing a certain parameterization of the generative process in terms of α_t [17].

Concluding the proof. Now for the final part of the proof, we substitute our value for α into the $\mathcal{I}_{\text{ISM}}$ loss from [17] which is equal to the negative of the evidence lower bound on $\mathbb{E}_{p_{\text{data}}(\mathbf{X}_0)}[\log p_0^{\theta}(\mathbf{X}_0)]$ up to a constant independent of θ. Defining $\beta_t(\mathbf{X}_t) = 1/\alpha_t(\mathbf{X}_t)$, we have

$$\mathcal{I}_{\text{ISM}}(\beta) = \int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\frac{\hat{\mathcal{L}}_t^* \beta_t(\mathbf{X}_t)}{\beta_t(\mathbf{X}_t)} + \hat{\mathcal{L}}_t \log \beta_t(\mathbf{X}_t)]\mathrm{d}t.$$

We split the spatial infintesimal generator of the forward process into the generator corresponding to the diffusion and the generator corresponding to the jump part, $\hat{\mathcal{L}} = \hat{\mathcal{L}}_t^{\text{diff}} + \hat{\mathcal{L}}_t^{\text{J}}$ with

$$\hat{\mathcal{L}}_t^{\text{diff}}(f)(\mathbf{X}) = \overrightarrow{\mathbf{b}}_t(\mathbf{X}) \cdot \nabla f(\mathbf{X}) + \tfrac{1}{2} g_t^2 \Delta f(\mathbf{X}).$$

and

$$\hat{\mathcal{L}}_t^{\text{J}}(f)(\mathbf{X}) = \overrightarrow{\lambda}_t(\mathbf{X}) \sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}.$$

By comparison with the approach to find the adjoint $\hat{\mathcal{K}}_t^*$, we also have $\hat{\mathcal{L}}_t^* = \hat{\mathcal{L}}_t^{\text{diff}*} + \hat{\mathcal{L}}_t^{\text{J}*}$ with

$$\hat{\mathcal{L}}_t^{\text{diff}*}(f)(\mathbf{X}) = -f(\mathbf{X}) \nabla \cdot \overrightarrow{\mathbf{b}}_t(\mathbf{X}) - \nabla f(\mathbf{X}) \cdot \overrightarrow{\mathbf{b}}_t(\mathbf{X}) + \tfrac{1}{2} g_t^2 \Delta f(\mathbf{X}).$$

In addition, we get

$$\hat{\mathcal{L}}_t^{\text{J}*}(f)(\mathbf{X}) = \sum_{m=1}^N \int_{\mathbf{y} \in \mathbb{R}^{md}} f(\mathbf{Y}) \overrightarrow{\lambda}_t(\mathbf{Y})(\overrightarrow{K}_t(\mathbf{X}|\mathbf{Y}) - \delta_{\mathbf{X}}(\mathbf{Y}))\mathrm{d}\mathbf{y}.$$

Finally, $\mathcal{I}_{\text{ISM}}$ becomes

$$\begin{aligned}\mathcal{I}_{\text{ISM}}(\beta) &= \int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\frac{\hat{\mathcal{L}}_t^{\text{diff}*} \beta_t(\mathbf{X}_t)}{\beta_t(\mathbf{X}_t)} + \hat{\mathcal{L}}_t^{\text{diff}} \log \beta_t(\mathbf{X}_t)]dt + \int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\frac{\hat{\mathcal{L}}_t^{\text{J}*} \beta_t(\mathbf{X}_t)}{\beta_t(\mathbf{X}_t)} + \hat{\mathcal{L}}^{\text{J}} \log \beta_t(\mathbf{X}_t)]\mathrm{d}t \\ &= \mathcal{I}_{\text{ISM}}^{\text{diff}}(\beta) + \mathcal{I}_{\text{ISM}}^{\text{J}}(\beta),\end{aligned}$$

where we have named the two terms corresponding to the diffusion and jump part of the process as $\mathcal{I}_{\text{ISM}}^{\text{diff}}$, $\mathcal{I}_{\text{ISM}}^{\text{J}}$ respectively. For the diffusion part of the loss, we use the denoising form of the objective proven in Appendix E of [17] which is equivalent to $\mathcal{I}_{\text{ISM}}^{\text{diff}}$ up to a constant independent of θ

$$\mathcal{I}_{\text{ISM}}^{\text{diff}}(\beta) = \int_0^T \mathbb{E}_{p_{0,t}(\mathbf{X}_0, \mathbf{X}_t)}[\frac{\hat{\mathcal{L}}_t^{\text{diff}}(p_{t|0}(\cdot|\mathbf{X}_0)\alpha_t(\cdot))(\mathbf{X}_t)}{p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\alpha_t(\mathbf{X}_t)} - \hat{\mathcal{L}}_t^{\text{diff}} \log(p_{t|0}(\cdot|\mathbf{X}_0)\alpha_t(\cdot))(\mathbf{X}_t)]\mathrm{d}t + \text{const.}$$

To simplify this expression, we first re-arrange $\hat{\mathcal{L}}_t^{\text{diff}}(h)$ for some general function $h : \mathcal{S} \to \mathbb{R}$.

$$\begin{aligned}\frac{\hat{\mathcal{L}}_t^{\text{diff}}(h)}{h} - \hat{\mathcal{L}}_t^{\text{diff}}(\log h) &= \tfrac{\overrightarrow{\mathbf{b}}_t \cdot \nabla h}{h} + \tfrac{1}{2} g_t^2 \tfrac{\Delta h}{h} - \overrightarrow{\mathbf{b}}_t \cdot \nabla \log h - \tfrac{1}{2} g_t^2 \Delta \log h \\ &= \tfrac{1}{2} g_t^2 (\tfrac{\nabla \cdot \nabla h}{h} - \nabla \cdot \nabla \log h) \\ &= \tfrac{1}{2} g_t^2 \|\nabla \log h\|^2.\end{aligned}$$

Setting $h = p_{t|0}(\cdot|\mathbf{X}_0)\alpha_t(\cdot)$, our diffusion part of the loss becomes

$$\mathcal{I}_{\text{ISM}}^{\text{diff}}(\beta) = \tfrac{1}{2} \int_0^T g_t^2 \mathbb{E}_{p_{0,t}(\mathbf{X}_0, \mathbf{X}_t)}[\|\nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) + \nabla \log \alpha_t(\mathbf{X}_t)\|^2]\mathrm{d}t + \text{const}$$

We then directly parameterize $\nabla \log \alpha_t(\mathbf{X}_t)$ as $-s_t^{\theta}(\mathbf{X}_t)$

$$\mathcal{I}_{\text{ISM}}^{\text{diff}}(\beta) = \tfrac{1}{2} \int_0^T g_t^2 \mathbb{E}_{p_{0,t}(\mathbf{X}_0, \mathbf{X}_t)}[\|\nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) - s_t^{\theta}(\mathbf{X}_t)\|^2]\mathrm{d}t + \text{const.}$$

We now focus on the expectation within the integral to re-write it in an easy to calculate form

$$\begin{aligned}&\mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}[\|\nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) - s_t^\theta(\mathbf{X}_t)\|^2]\\&= \mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}[\|s_t^\theta(\mathbf{X}_t)\|^2 - 2s_t^\theta(\mathbf{X}_t)^T\nabla \log p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)] + \text{const}\end{aligned}$$

Now we note that we can re-write $\nabla \log p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)$ using M_t where M_t is a mask variable $M_t \in \{0,1\}^{n_0}$ that is 0 for components of $\mathbf{X}_0$ that have been deleted to get to $\mathbf{X}_t$ and 1 for components that remain in $\mathbf{X}_t$.

$$\begin{aligned}\nabla \log p_{0,t}(\mathbf{X}_0,\mathbf{X}_t) &= \tfrac{1}{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}\nabla p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)\\&= \tfrac{1}{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}\nabla \textstyle\sum_{M_t} p_{0,t}(\mathbf{X}_0,\mathbf{X}_t,M_t)\\&= \textstyle\sum_{M_t} \tfrac{1}{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}\nabla p_{0,t}(\mathbf{X}_0,\mathbf{X}_t,M_t)\\&= \textstyle\sum_{M_t} \tfrac{p(n_t,M_t,\mathbf{X}_0)}{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}\nabla p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t)\\&= \textstyle\sum_{M_t} \tfrac{p(M_t|\mathbf{X}_0,\mathbf{X}_t)}{p(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t)}\nabla p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t)\\&= \mathbb{E}_{p(M_t|\mathbf{X}_0,\mathbf{X}_t)}[\nabla \log p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t)]\end{aligned}$$

Substituting this back in we get

$$\begin{aligned}&\mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}[\|\nabla \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0) - s_t^\theta(\mathbf{X}_t)\|^2]\\&= \mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t)}[\|s_t^\theta(\mathbf{X}_t)\|^2 - 2s_t^\theta(\mathbf{X}_t)^T\mathbb{E}_{p(M_t|\mathbf{X}_0,\mathbf{X}_t)}[\nabla \log p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t)]] + \text{const}\\&= \mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t,M_t)}[\|\nabla \log p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t) - s_t^\theta(\mathbf{X}_t)\|^2] + \text{const.}\end{aligned}$$

Therefore, the diffusion part of $\mathcal{I}_{\text{ISM}}$ can be written as

$$\mathcal{I}_{\text{ISM}}^{\text{diff}}(\beta) = \tfrac{T}{2}\mathbb{E}_{\mathcal{U}(t;0,T)p_{0,t}(\mathbf{X}_0,\mathbf{X}_t,M_t)}[g_t^2\|\nabla \log p_{t|0}(\mathbf{x}_t|n_t,\mathbf{X}_0,M_t) - s_t^\theta(\mathbf{X}_t)\|^2] + \text{const.}$$

We now focus on the jump part of the loss $\mathcal{I}_{\text{ISM}}^{\text{J}}$. We first substitute in $\hat{\mathcal{L}}_t^{\text{J}}$ and $\hat{\mathcal{L}}_t^{\text{J}*}$

$$\begin{aligned}\mathcal{I}_{\text{ISM}}^{\text{J}} = \textstyle\int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\sum_m \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{\lambda}_t(\mathbf{Y})\tfrac{\beta_t(\mathbf{Y})}{\beta_t(\mathbf{X}_t)}(\overrightarrow{K}_t(\mathbf{X}_t|\mathbf{Y}) - \delta_{\mathbf{Y}}(\mathbf{X}_t))\mathrm{d}\mathbf{y} +\\ \overrightarrow{\lambda}_t(\mathbf{X}_t)\textstyle\sum_{m=1}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)\log\beta_t(\mathbf{Y})\mathrm{d}\mathbf{y} - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log\beta_t(\mathbf{X}_t)]\mathrm{d}t.\end{aligned} \tag{18}$$

Noting that $\beta_t(\mathbf{X}_t) = 1/\alpha_t(\mathbf{X}_t)$, we get

$$\tfrac{\beta_t(\mathbf{X}_t)}{\beta_t(\mathbf{Y})} = \tfrac{\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{Y})\overleftarrow{K}^\theta_{T-t}(\mathbf{X}_t|\mathbf{Y})}{\overrightarrow{\lambda}_t(\mathbf{X}_t)\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)} \quad \text{for } n_t \neq m \tag{19}$$

or swapping labels for $\mathbf{X}_t$ and $\mathbf{Y}$,

$$\tfrac{\beta_t(\mathbf{Y})}{\beta_t(\mathbf{X}_t)} = \tfrac{\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{X}_t)\overleftarrow{K}^\theta_{T-t}(\mathbf{Y}|\mathbf{X}_t)}{\overrightarrow{\lambda}_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{X}_t|\mathbf{Y})} \quad \text{for } n_t \neq m \tag{20}$$

Substituting (19) into the second line and (20) into the first line of (18) and using the fact that $\overrightarrow{K}_t(\mathbf{X}_t|\mathbf{Y}) = 0$ for $n_t = m$, we obtain

$$\begin{aligned}\mathcal{I}_{\text{ISM}}^{\text{J}} = \textstyle\int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\sum_{m=1\setminus n_t}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{\lambda}_t(\mathbf{Y})\tfrac{\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{X}_t)\overleftarrow{K}^\theta_{T-t}(\mathbf{Y}|\mathbf{X}_t)}{\overrightarrow{\lambda}_t(\mathbf{Y})\overrightarrow{K}_t(\mathbf{X}_t|\mathbf{Y})}\overrightarrow{K}_t(\mathbf{X}_t|\mathbf{Y})\mathrm{d}\mathbf{y}\\ - \textstyle\sum_{m=1}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{\lambda}_t(\mathbf{Y})\tfrac{\beta_t(\mathbf{Y})}{\beta_t(\mathbf{X}_t)}\delta_{\mathbf{Y}}(\mathbf{X}_t)\mathrm{d}\mathbf{y}\\ + \overrightarrow{\lambda}_t(\mathbf{X}_t)\textstyle\sum_{m=1\setminus n_t}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)\{\log\beta_t(\mathbf{X}_t) - \log\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{Y})\\ - \log\overleftarrow{K}^\theta_{T-t}(\mathbf{X}_t|\mathbf{Y}) + \log\overrightarrow{\lambda}_t(\mathbf{X}_t) + \log\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)\}\mathrm{d}\mathbf{y}\\ - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log\beta_t(\mathbf{X}_t)]\mathrm{d}t.\end{aligned}$$

Hence, we have

$$\begin{aligned}\mathcal{I}_{\text{ISM}}^{\text{J}} = \textstyle\int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{X}_t)\sum_{m=1\setminus n_t}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overleftarrow{K}^\theta_{T-t}(\mathbf{Y}|\mathbf{X}_t)\mathrm{d}\mathbf{y} - \overrightarrow{\lambda}_t(\mathbf{X}_t)\tfrac{\beta_t(\mathbf{X}_t)}{\beta_t(\mathbf{X}_t)} +\\ \overrightarrow{\lambda}_t(\mathbf{X}_t)\textstyle\sum_{m=1\setminus n_t}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)\{-\log\overleftarrow{\lambda}^\theta_{T-t}(\mathbf{Y}) - \log\overleftarrow{K}^\theta_{T-t}(\mathbf{X}_t|\mathbf{Y})\}\mathrm{d}\mathbf{y}]\mathrm{d}t + \text{const.}\end{aligned}$$

This can be rewritten as
$\mathcal{I}^{\text{J}}_{\text{ISM}} = \int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)}[\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) + \overrightarrow{\lambda}_t(\mathbf{X}_t)\mathbb{E}_{\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)}[-\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) - \log \overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}_t|\mathbf{Y})]]\mathrm{d}t + \text{const.}$
Therefore, we have
$\mathcal{I}^{\text{J}}_{\text{ISM}} = T\mathbb{E}_{\mathcal{U}(t;0,T)p_t(\mathbf{X}_t)\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t)}[\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}_t|\mathbf{Y})] + \text{const.}$
Finally, using the definition of the forward and backward kernels, i.e. $\overrightarrow{K}_t(\mathbf{Y}|\mathbf{X}_t) = \sum_{i=1}^n K^{\text{del}}(i|n)\delta_{\text{del}(\mathbf{X},i)}(\mathbf{Y})$ and $\overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}_t|\mathbf{Y}) = \int_{\mathbf{x}^{\text{add}}} \sum_{i=1}^n A^{\theta}_t(\mathbf{x}^{\text{add}}, i|\mathbf{Y})\delta_{\text{ins}(\mathbf{Y},\mathbf{x}^{\text{add}},i)}(\mathbf{X}_t)\mathrm{d}\mathbf{x}^{\text{add}}$, we get
$\mathcal{I}^{\text{J}}_{\text{ISM}} = T\mathbb{E}_{\mathcal{U}(t;0,T)p_t(\mathbf{X}_t)}[\sum_{m=1}^N \int_{\mathbf{y}\in\mathbb{R}^{md}} \sum_{i=1}^{n_t} K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})(\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y})$
$- \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}_t|\mathbf{Y}))\mathrm{d}\mathbf{y}] + \text{const}$
We get
$\mathcal{I}^{\text{J}}_{\text{ISM}} = T\mathbb{E}_{\mathcal{U}(t;0,T)p_t(\mathbf{X}_t)K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})}$
$[\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{K}^{\theta}_{T-t}(\mathbf{X}_t|\mathbf{Y})] + \text{const.}$
Therefore, we have
$\mathcal{I}^{\text{J}}_{\text{ISM}} = T\mathbb{E}_{\mathcal{U}(t;0,T)p_t(\mathbf{X}_t)K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})}$
$[\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log A^{\theta}_{T-t}(\mathbf{x}^{\text{add}}, i|\mathbf{Y})] + \text{const.}$
Putting are expressions for $\mathcal{I}^{\text{diff}}_{\text{ISM}}$ and $\mathcal{I}^{\text{J}}_{\text{ISM}}$ together we obtain
$\mathcal{I}_{\text{ISM}} = \frac{T}{2}\mathbb{E}[g_t^2\|\nabla \log p_{t|0}(\mathbf{x}_t|n_t, \mathbf{X}_0, M_t) - s^{\theta}_t(\mathbf{X}_t)\|^2] +$
$T\mathbb{E}[\overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{X}_t) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log \overleftarrow{\lambda}^{\theta}_{T-t}(\mathbf{Y}) - \overrightarrow{\lambda}_t(\mathbf{X}_t)\log A^{\theta}_{T-t}(\mathbf{x}^{\text{add}}, i|\mathbf{Y})] + \text{const.}$
We get that $-\mathcal{I}_{\text{ISM}}$ gives us our evidence lower bound on $\mathbb{E}_{p_{\text{data}}(\mathbf{X}_0)}[\log p^{\theta}_0(\mathbf{X}_0)]$ up to a constant that does not depend on θ. In the main text we have used a time notation such that the backward process runs backwards from $t = T$ to $t = 0$. To align with the notation of time used in the main text we change $T - t$ to t on subscripts for $\overleftarrow{\lambda}^{\theta}_{T-t}$ and A^{θ}_{T-t}. We also will use the fact that $\overrightarrow{\lambda}_t(\mathbf{X}_t)$ depends only on the number of components in $\mathbf{X}_t$, $\overrightarrow{\lambda}_t(\mathbf{X}_t) = \overrightarrow{\lambda}_t(n_t)$.
$\mathcal{L}(\theta) = -\frac{T}{2}\mathbb{E}[g_t^2\|s^{\theta}_t(\mathbf{X}_t) - \nabla_{\mathbf{x}_t}\log p_{t|0}(\mathbf{x}_t|n_t, \mathbf{X}_0, M_t)\|^2] +$
$T\mathbb{E}[-\overleftarrow{\lambda}^{\theta}_t(\mathbf{X}_t) + \overrightarrow{\lambda}_t(n_t)\log \overleftarrow{\lambda}^{\theta}_t(\mathbf{Y}) + \overrightarrow{\lambda}_t(n_t)\log A^{\theta}_t(\mathbf{x}^{\text{add}}, i|\mathbf{Y})] + \text{const.}$

Tightness of the lower bound Now that we have derived the ELBO as in Proposition 2, we show that the maximizers of the ELBO are tight, i.e. that they close the variational gap. We do this by proving the general ELBO presented in [17] has this property and therefore ours, which is a special case of this general ELBO, also has that the optimum parameters close the variational gap.

To state our proposition, we recall the setting of [17]. The forward noising process is denoted $(\mathbf{Y}_t)_{t\geq 0}$ and associated with an infinitesimal generator $\hat{\mathcal{L}}$ its extension $(t, \mathbf{Y}_t)_{t\geq 0}$ is associated with the infinitesimal generator $\mathcal{L}$, i.e. $\mathcal{L} = \partial_t + \hat{\mathcal{L}}$. We also define the score-matching operator Φ given for any f for which it is defined by
$$\Phi(f) = \mathcal{L}(f)/f - \mathcal{L}(\log(f)).$$
We recall that according to [17, Equation (8)] and under [17, Assumption 1, Assumption2], we have
$$\log p_T(\mathbf{Y}_0) \geq \mathbb{E}[\log p_0(\mathbf{Y}_T) - \int_0^T \mathcal{L}(v/\beta)/(v/\beta) + \mathcal{L}(\log\beta)\mathrm{d}t],$$
with $v_t = p_{T-t}$ for any $t \in [0, T]$. We define the *variational gap* Gap as follows
$$\text{Gap} = \mathbb{E}[\log p_T(\mathbf{Y}_0) - \log p_0(\mathbf{Y}_T) + \int_0^T \mathcal{L}(v/\beta)/(v/\beta) + \mathcal{L}(\log\beta)\mathrm{d}t].$$
In addition, using Itô Formula, we have that $\log v_T(\mathbf{Y}_T) - \log v_0(\mathbf{Y}_0) = \int_0^T \mathcal{L}(v)\mathrm{d}t$. Assuming that $\mathbb{E}[|\log v_T(\mathbf{Y}_T) - \log v_0(\mathbf{Y}_0)|] < +\infty$, we get
$$\text{Gap} = \mathbb{E}[\int_0^T -\mathcal{L}(\log v) + \mathcal{L}(v/\beta)/(v/\beta) + \mathcal{L}(\log\beta)\mathrm{d}t] = \mathbb{E}[\int_0^T \Phi(v/\beta)\mathrm{d}t].$$
In particular, using [17, Proposition 1], we get that $\text{Gap} \geq 0$ and $\text{Gap} = 0$ if and only if $\beta \propto v$. In addition, the ELBO is maximized if and only if $\beta \propto v$, see [17, Equation 10] and the remark that follows. Therefore, we have that: if we maximize the ELBO then the ELBO is tight. Combining this with the fact that the ELBO is maximized at the time reversal [17], then we have that when our jump diffusion parameters match the time reversal, our variational gap is 0.

Other approaches. Another way to derive the ELBO is to follow the steps of [15] directly, since [17] is a general framework extending this approach. The key formulae to derive the result and the ELBO is 1) a Feynman-Kac formula 2) a Girsanov formula. In the case of jump diffusions (with jump in $\mathbb{R}^d$) a Girsanov formula has been established by [22]. Extending this result to one-point compactification space would allow us to prove directly Proposition 2 without having to rely on the general framework of [17].

A.4 Proof of Proposition 3

We start by recalling the form for the time reversal given in Proposition 1

$$\overleftarrow{\lambda}^*_t(\mathbf{X}) = \overrightarrow{\lambda}_t(n+1)\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i))\mathrm{d}\mathbf{y}^{\text{add}}/p_t(\mathbf{X}).$$

We then introduce a marginalization over $\mathbf{X}_0$

$$\begin{aligned}
\overleftarrow{\lambda}^*_t(\mathbf{X}) &= \overrightarrow{\lambda}_t(n+1)\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}}\sum_{n_0}\int_{\mathbf{x}_0} p_{0,t}(\mathbf{X}_0,\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i))\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{y}^{\text{add}}/p_t(\mathbf{X}) \\
&= \overrightarrow{\lambda}_t(n+1)\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}}\sum_{n_0}\int_{\mathbf{x}_0} \frac{p_0(\mathbf{X}_0)}{p_t(\mathbf{X})} p_{t|0}(\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{y}^{\text{add}} \\
&= \overrightarrow{\lambda}_t(n+1)\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}}\sum_{n_0}\int_{\mathbf{x}_0} \frac{p_{0|t}(\mathbf{X}_0|\mathbf{X})}{p_{t|0}(\mathbf{X}|\mathbf{X}_0)} p_{t|0}(\text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{y}^{\text{add}} \\
&= \overrightarrow{\lambda}_t(n+1)\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}}\sum_{n_0}\int_{\mathbf{x}_0} \frac{p_{0|t}(n_0|\mathbf{X})p_{0|t}(\mathbf{x}_0|\mathbf{X},n_0)}{p_{t|0}(n|\mathbf{X}_0)p_{t|0}(\mathbf{x}|\mathbf{X}_0,n)}\times \\
&\qquad\qquad p_{t|0}(n+1|\mathbf{X}_0)p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1)\mathrm{d}\mathbf{x}_0\mathrm{d}\mathbf{y}^{\text{add}}
\end{aligned}$$

where $(n+1,\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)) = \text{ins}(\mathbf{X},\mathbf{y}^{\text{add}},i)$. Now using the fact the forward component deletion process does not depend on $\mathbf{x}_0$, only n_0, we have $p_{t|0}(n|\mathbf{X}_0) = p_{t|0}(n|n_0)$ and $p_{t|0}(n+1|\mathbf{X}_0) = p_{t|0}(n+1|n_0)$. Using this result, we get

$$\begin{aligned}
\overleftarrow{\lambda}^*_t(\mathbf{X}) = &\overrightarrow{\lambda}_t(n+1)\sum_{n_0}\{\frac{p_{t|0}(n+1|n_0)}{p_{t|0}(n|n_0)} p_{0|t}(n_0|\mathbf{X})\times \\
&\int_{\mathbf{x}_0} \frac{\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1)\mathrm{d}\mathbf{y}^{\text{add}}}{p_{t|0}(\mathbf{x}|\mathbf{X}_0,n)} p_{0|t}(\mathbf{x}_0|\mathbf{X},n_0)\mathrm{d}\mathbf{x}_0\}. \qquad (21)
\end{aligned}$$

We now focus on the probability ratio within the integral over $\mathbf{x}_0$. We will show that this ratio is 1. We start with the numerator, introducing a marginalization over possible mask variables between $\mathbf{X}_0$ and $(n+1,\mathbf{z})$, denoted $M^{(n+1)}$ with $M^{(n+1)}$ having $n+1$ ones and $n_0-(n+1)$ zeros.

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1)\mathrm{d}\mathbf{y}^{\text{add}} \\
&= \textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\sum_{M^{(n+1)}}\int_{\mathbf{y}^{\text{add}}} p_{t|0}(M^{(n+1)},\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1)\mathrm{d}\mathbf{y}^{\text{add}} \\
&= \textstyle\sum_{M^{(n+1)}}\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)p_{t|0}(M^{(n+1)}|\mathbf{X}_0,n+1)\int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1,M^{(n+1)})\mathrm{d}\mathbf{y}^{\text{add}}
\end{aligned}$$

Now, for our forward process we have

$$p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1,M^{(n+1)}) = \textstyle\prod_{j=1}^{n+1}\mathcal{N}(\mathbf{z}^{(j)};\sqrt{\alpha_t}M^{(n+1)}(\mathbf{X}_0)^j,(1-\alpha_t)I_d)$$

where $\mathbf{z}$ is shorthand for $\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)$, $\mathbf{z}^{(j)}$ is the vector in $\mathbb{R}^d$ for the jth component of $\mathbf{z}$ and $M^{(n+1)}(\mathbf{X}_0)^j$ is the vector in $\mathbb{R}^d$ corresponding to the component in $\mathbf{X}_0$ corresponding to the jth one in the $M^{(n+1)}$ mask. Integrating out $\mathbf{y}^{\text{add}}$ we have

$$\textstyle\int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1,M^{(n+1)})\mathrm{d}\mathbf{y}^{\text{add}} = \prod_{j=1}^{n}\mathcal{N}(\mathbf{x}^{(j)};\sqrt{\alpha_t}M^{(n+1)\setminus i}(\mathbf{X}_0)^j,(1-\alpha_t)I_d),$$

where $M^{(n+1)\setminus i}$ denotes a mask variable obtained by setting the ith one of $M^{(n+1)}$ to zero. Hence, we have

$$\begin{aligned}
&\textstyle\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)\int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X},\mathbf{y}^{\text{add}},i)|\mathbf{X}_0,n+1)\mathrm{d}\mathbf{y}^{\text{add}} \\
&= \textstyle\sum_{M^{(n+1)}}\sum_{i=1}^{n+1} K^{\text{del}}(i|n+1)p_{t|0}(M^{(n+1)}|\mathbf{X}_0,n+1) \\
&\quad \textstyle\prod_{j=1}^{n}\mathcal{N}(\mathbf{x}^{(j)};\sqrt{\alpha_t}M^{(n+1)\setminus i}(\mathbf{X}_0)^j,(1-\alpha_t)I_d). \qquad (22)
\end{aligned}$$

We now re-write the denominator from (21) introducing a marginalization over mask variables, $M^{(n)}$

$$p_{t|0}(\mathbf{x}|\mathbf{X}_0,n) = \textstyle\sum_{M^{(n)}} p_{t|0}(M^{(n)}|\mathbf{X}_0,n)p_{t|0}(\mathbf{x}|M^{(n)},\mathbf{X}_0,n). \qquad (23)$$

We use the following recursion for the probabilities assigned to mask variables

$$p_{t|0}(M^{(n)}|\mathbf{X}_0, n) = \sum_{M^{(n+1)}} \sum_{i=1}^{n+1} \mathbb{I}\{M^{(n+1)\setminus i} = M^{(n)}\} K^{\text{del}}(i|n+1) p_{t|0}(M^{(n+1)}|\mathbf{X}_0, n+1).$$

Substituting this into (23) gives

$$\begin{aligned}
p_{t|0}(\mathbf{x}|\mathbf{X}_0, n) &= \sum_{M^{(n)}} \sum_{M^{(n+1)}} \sum_{i=1}^{n+1} \mathbb{I}\{M^{(n+1)\setminus i} = M^{(n)}\} K^{\text{del}}(i|n+1) \times \\
&\qquad p_{t|0}(M^{(n+1)}|\mathbf{X}_0, n+1) p_{t|0}(\mathbf{x}|M^{(n)}, \mathbf{X}_0, n) \\
&= \sum_{M^{(n)}} \sum_{M^{(n+1)}} \sum_{i=1}^{n+1} \mathbb{I}\{M^{(n+1)\setminus i} = M^{(n)}\} K^{\text{del}}(i|n+1) \\
&\qquad \times p_{t|0}(M^{(n+1)}|\mathbf{X}_0, n+1) \prod_{j=1}^{n} \mathcal{N}(\mathbf{x}^{(j)}; \sqrt{\alpha_t} M^{(n)}(\mathbf{X}_0)^j, (1-\alpha_t) I_d) \\
&= \sum_{M^{(n+1)}} \sum_{i=1}^{n+1} K^{\text{del}}(i|n+1) p_{t|0}(M^{(n+1)}|\mathbf{X}_0, n+1) \\
&\qquad \times \prod_{j=1}^{n} \mathcal{N}(\mathbf{x}^{(j)}; \sqrt{\alpha_t} M^{(n+1)\setminus i}(\mathbf{X}_0)^j, (1-\alpha_t) I_d).
\end{aligned}$$

By comparing with (22), we can see that

$$p_{t|0}(\mathbf{x}|\mathbf{X}_0, n) = \sum_{i=1}^{n+1} K^{\text{del}}(i|n+1) \int_{\mathbf{y}^{\text{add}}} p_{t|0}(\mathbf{z}(\mathbf{X}, \mathbf{y}^{\text{add}}, i)|\mathbf{X}_0, n+1) \mathrm{d}\mathbf{y}^{\text{add}}.$$

This shows that the probability ratio in (21) is 1. Therefore, we have

$$\begin{aligned}
\overleftarrow{\lambda}^*_t(\mathbf{X}) &= \overrightarrow{\lambda}_t(n+1) \sum_{n_0} \{ \tfrac{p_{t|0}(n+1|n_0)}{p_{t|0}(n|n_0)} p_{0|t}(n_0|\mathbf{X}) \int_{\mathbf{x}_0} p_{0|t}(\mathbf{x}_0|\mathbf{X}, n_0) \mathrm{d}\mathbf{x}_0 \} \\
&= \overrightarrow{\lambda}_t(n+1) \sum_{n_0} \tfrac{p_{t|0}(n+1|n_0)}{p_{t|0}(n|n_0)} p_{0|t}(n_0|\mathbf{X}),
\end{aligned}$$

which concludes the proof.

$p_{t|0}(n|n_0)$ can be analytically calculated when $\overrightarrow{\lambda}_t(n)$ is of a simple enough form. When $\overrightarrow{\lambda}_t(n)$ does not depend on n then the dimension deletion process simply becomes a time inhomogeneous Poisson process. Therefore, we would have

$$p_{t|0}(n|n_0) = \tfrac{(\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s)^{n_0-n}}{(n_0-n)!} \exp(-\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s).$$

In our experiments we set $\overrightarrow{\lambda}_t(n=1) = 0$ to stop the dimension deletion process when we reach a single component. If we have $\overrightarrow{\lambda}_t(n) = \overrightarrow{\lambda}_t(m)$ for all $n, m > 1$ then we can still use the time inhomogeneous Poisson process formula for $n > 1$ and find the probability for $n = 1$, $p_{t|0}(n=1|n_0)$ by requiring $p_{t|0}(n|n_0)$ to be a valid normalized distribution. Therefore, for the case that $\overrightarrow{\lambda}_t(n) = \overrightarrow{\lambda}_t(m)$ for all $n, m > 1$ and $\overrightarrow{\lambda}_t(n=1) = 0$, we have

$$p_{t|0}(n|n_0) = \begin{cases} \tfrac{(\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s)^{n_0-n}}{(n_0-n)!} \exp(-\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s) & 1 < n \leq n_0 \\ 1 - \sum_{m=2}^{n_0} \tfrac{(\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s)^{n_0-m}}{(n_0-m)!} \exp(-\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s) & n = 1 \end{cases}$$

In cases where $\overrightarrow{\lambda}_t(n)$ depends on n not just for $n = 1$, $p_{t|0}(n|n_0)$ can become more difficult to calculate analytically. However, since the probability distributions are all 1-dimensional over n, it is very cheap to simply simulate the forward dimension deletion process many times and empirically estimate $p_{t|0}(n|n_0)$ although we do not need to do this for our experiments.

A.5 The Objective is Maximized at the Time Reversal

In this section, we analyze the objective $\mathcal{L}(\theta)$ as a standalone object and determine the optimum values for s^θ_t, $\overleftarrow{\lambda}^\theta_t$ and A^θ_t directly. This is in order to gain intuition directly into the learning signal of $\mathcal{L}(\theta)$ without needing to refer to stochastic process theory.

The definition of $\mathcal{L}(\theta)$ as in the main text is

$$\begin{aligned}
\mathcal{L}(\theta) = &-\tfrac{T}{2} \mathbb{E}[g_t^2 \| s^\theta_t(\mathbf{X}_t) - \nabla_{\mathbf{x}_t} \log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0, n_t, M_t) \|^2] + \\
&T \mathbb{E}[- \overleftarrow{\lambda}^\theta_t(\mathbf{X}_t) + \overrightarrow{\lambda}_t(n_t) \log \overleftarrow{\lambda}^\theta_t(\mathbf{Y}) + \overrightarrow{\lambda}_t(n_t) \log A^\theta_t(\mathbf{x}^{\text{add}}_t, i|\mathbf{Y})] + C.
\end{aligned}$$

with the expectations taken over $\mathcal{U}(t; 0, T) p_{0,t}(\mathbf{X}_0, \mathbf{X}_t, M_t) K^{\text{del}}(i|n_t) \delta_{\text{del}(\mathbf{X}_t, i)}(\mathbf{Y})$.

Continuous optimum. We start by analysing the objective for s_t^θ. This part of $\mathcal{L}(\theta)$ can be written as

$$-\tfrac{1}{2}\int_0^T g_t^2 \mathbb{E}_{p_{0,t}(\mathbf{X}_0,\mathbf{X}_t,M_t)}[\|s_t^\theta(\mathbf{X}_t) - \nabla_{\mathbf{x}_t}\log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0,n_t,M_t)\|^2]\mathrm{d}t$$

We now use the fact that the function that minimizes an L_2 regression problem $\min_f \mathbb{E}_{p(x,y)}[\|f(x) - y\|^2]$ is the conditional expectation of the target $f^*(x) = \mathbb{E}_{p(y|x)}[y]$. Therefore the optimum value for $s_t^\theta(\mathbf{X}_t)$ is

$$\begin{aligned}
s_t^*(\mathbf{X}_t) &= \mathbb{E}_{p(M_t,\mathbf{X}_0|\mathbf{X}_t)}[\nabla_{\mathbf{x}_t}\log p_{t|0}(\mathbf{x}_t|\mathbf{X}_0,n_t,M_t)] \\
&= \textstyle\sum_{M_t}\sum_{n_0=1}^N \int_{\mathbf{x}_0\in\mathbb{R}^{n_0 d}} p(M_t,n_0,\mathbf{x}_0|\mathbf{X}_t)\nabla_{\mathbf{x}_t}\log p_{t|0}(\mathbf{x}_t|\mathbf{x}_0,n_0,n_t,M_t)\mathrm{d}\mathbf{x}_0 \\
&= \textstyle\sum_{M_t}\sum_{n_0=1}^N \int_{\mathbf{x}_0\in\mathbb{R}^{n_0 d}} \frac{p(M_t,n_0,\mathbf{x}_0|\mathbf{X}_t)}{p_{t|0}(\mathbf{x}_t|\mathbf{x}_0,n_0,n_t,M_t)}\nabla_{\mathbf{x}_t} p_{t|0}(\mathbf{x}_t|\mathbf{x}_0,n_0,n_t,M_t)\mathrm{d}\mathbf{x}_0 \\
&= \textstyle\sum_{M_t}\sum_{n_0=1}^N \int_{\mathbf{x}_0\in\mathbb{R}^{n_0 d}} \frac{p(\mathbf{x}_0,n_0,n_t,M_t)}{p(n_t,\mathbf{x}_t)}\nabla_{\mathbf{x}_t} p_{t|0}(\mathbf{x}_t|\mathbf{x}_0,n_0,n_t,M_t)\mathrm{d}\mathbf{x}_0 \\
&= \tfrac{1}{p(n_t,\mathbf{x}_t)}\textstyle\sum_{M_t}\sum_{n_0=1}^N \int_{\mathbf{x}_0\in\mathbb{R}^{n_0 d}} \nabla_{\mathbf{x}_t} p(\mathbf{x}_t,\mathbf{x}_0,n_0,n_t,M_t)\mathrm{d}\mathbf{x}_0 \\
&= \tfrac{1}{p(n_t,\mathbf{x}_t)}\nabla_{\mathbf{x}_t}\textstyle\sum_{M_t}\sum_{n_0=1}^N \int_{\mathbf{x}_0\in\mathbb{R}^{n_0 d}} p(\mathbf{x}_t,\mathbf{x}_0,n_0,n_t,M_t)\mathrm{d}\mathbf{x}_0 \\
&= \tfrac{1}{p(n_t,\mathbf{x}_t)}\nabla_{\mathbf{x}_t} p(\mathbf{x}_t,n_t) = \nabla_{\mathbf{x}_t}\log p(\mathbf{X}_t).
\end{aligned}$$

Therefore, the optimum value for $s_t^\theta(\mathbf{X}_t)$ is $\nabla_{\mathbf{x}_t}\log p(\mathbf{X}_t)$ which is the value that gives $\overleftarrow{\mathbf{b}}_t$ to be the time reversal of $\overrightarrow{\mathbf{b}}_t$ as stated in Proposition 1.

Jump rate optimum. The learning signal for $\overleftarrow{\lambda}_t^\theta$ comes from these two terms in $\mathcal{L}(\theta)$

$$T\mathbb{E}[-\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) + \overrightarrow{\lambda}_t(n_t)\log\overleftarrow{\lambda}_t^\theta(\mathbf{Y})] \tag{24}$$

This expectation is maximized when for each test input $\mathbf{Z}$ and test time t, we have the following expression maximized

$$-p_t(\mathbf{Z})\overleftarrow{\lambda}_t^\theta(\mathbf{Z}) + \textstyle\sum_{i=}^{n_z+1}\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{Z},\mathbf{y}^{\text{add}},i))K^{\text{del}}(i|n_z+1)\mathrm{d}\mathbf{y}^{\text{add}} \times \overrightarrow{\lambda}_t(n_z+1)\log\overleftarrow{\lambda}_t^\theta(\mathbf{Z}),$$

because $p_t(\mathbf{Z})$ is the probability $\mathbf{Z}$ gets drawn as a full sample from the forward process and $\sum_{i=}^{n_z+1}\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{Z},\mathbf{y}^{\text{add}},i))K^{\text{del}}(i|n_z+1)\mathrm{d}\mathbf{y}^{\text{add}}$ is the probability that a sample one component bigger than $\mathbf{Z}$ gets drawn from the forward process and then a component is deleted to get to $\mathbf{Z}$. Therefore the first probability is the probability that test input $\mathbf{Z}$ and test time t appear as the first term in (24) whereas the second probability is the probability that test input $\mathbf{Z}$ and test time t appear as the second term in (24).

We now use the fact that, for constants b and c,

$$\text{argmax}_a \quad -ba + c\log a = \tfrac{c}{b}.$$

We therefore have the optimum $\overleftarrow{\lambda}_t^\theta(\mathbf{Z})$ as

$$\overleftarrow{\lambda}_t^*(\mathbf{Z}) = \overrightarrow{\lambda}_t(n_z+1)\frac{\sum_{i=}^{n_z+1}\int_{\mathbf{y}^{\text{add}}} p_t(\text{ins}(\mathbf{Z},\mathbf{y}^{\text{add}},i))K^{\text{del}}(i|n_z+1)\mathrm{d}\mathbf{y}^{\text{add}}}{p_t(\mathbf{Z})}$$

which is the form for the time-reversal given in Proposition (1).

Jump kernel optimum. Finally, we analyse the part of $\mathcal{L}(\theta)$ for learning $A_t^\theta(\mathbf{x}_t^{\text{add}},i|\mathbf{Y})$,

$$\begin{aligned}
&T\mathbb{E}[\overrightarrow{\lambda}_t(n_t)\log A_t^\theta(\mathbf{x}_t^{\text{add}},i|\mathbf{Y})] \\
&= \textstyle\int_0^T \mathbb{E}_{p_t(\mathbf{X}_t)K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})}[\overrightarrow{\lambda}_t(n_t)\log A_t^\theta(\mathbf{x}_t^{\text{add}},i|\mathbf{Y})]\mathrm{d}t \\
&= \textstyle\int_0^T \mathbb{E}_{p_t(n_t)}[\overrightarrow{\lambda}_t(n_t)\mathbb{E}_{p_t(\mathbf{x}_t|n_t)K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})}[\log A_t^\theta(\mathbf{x}_t^{\text{add}},i|\mathbf{Y})]]\mathrm{d}t.
\end{aligned}$$

We now re-write the joint probability distribution that the inner expectation is taken with respect to,

$$p_t(\mathbf{x}_t|n_t)K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y}) = \tilde{p}(\mathbf{Y}|n_t)p(\mathbf{x}_t^{\text{add}},i|\mathbf{Y})\delta_{\mathbf{y}}(\mathbf{x}_t^{\text{base}}).$$

with

$$\tilde{p}(\mathbf{Y}|n_t) = \sum_{i=1}^{n_t} \int_{\mathbf{x}_t} p_t(\mathbf{x}_t|n_t) K^{\text{del}}(i|n_t) \delta_{\text{del}(\mathbf{x}_t,i)}(\mathbf{Y}) \mathrm{d}\mathbf{x}_t,$$

and

$$p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \propto p_t(\mathbf{x}_t|n_t) K^{\text{del}}(i|n_t),$$

and $\mathbf{x}_t^{\text{base}} \in \mathbb{R}^{(n_t-1)d}$ referring to the $n_t - 1$ components of $\mathbf{x}_t$, that are not $\mathbf{x}_t^{\text{add}}$ i.e. $\mathbf{X}_t = \text{ins}((\mathbf{x}_t^{\text{base}}, n_t - 1), \mathbf{x}_t^{\text{add}}, i)$. We then have

$$\begin{aligned}
&T\mathbb{E}[\overrightarrow{\lambda}_t(n_t) \log A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})] \\
&= \int_0^T \mathbb{E}_{p_t(n_t)}[\overrightarrow{\lambda}_t(n_t) \mathbb{E}_{\tilde{p}(\mathbf{Y}|n_t) p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \delta_{\mathbf{y}}(\mathbf{x}_t^{\text{base}})}[\log A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})]] \mathrm{d}t \\
&= \int_0^T \mathbb{E}_{p_t(n_t)}[\overrightarrow{\lambda}_t(n_t) \mathbb{E}_{\tilde{p}(\mathbf{Y}|n_t) p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \delta_{\mathbf{y}}(\mathbf{x}_t^{\text{base}})}[\log A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})]] \mathrm{d}t \\
&\quad - \int_0^T \mathbb{E}_{p_t(n_t)}[\overrightarrow{\lambda}_t(n_t) \mathbb{E}_{\tilde{p}(\mathbf{Y}|n_t) p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \delta_{\mathbf{y}}(\mathbf{x}_t^{\text{base}})}[\log p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})]] \mathrm{d}t + \text{const} \\
&= \int_0^T \mathbb{E}_{p_t(n_t)}[\overrightarrow{\lambda}_t(n_t) \mathbb{E}_{\tilde{p}(\mathbf{Y}|n_t) \delta_{\mathbf{y}}(\mathbf{x}_t^{\text{base}})}[-\text{KL}(p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \,||\, A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})]] \mathrm{d}t + \text{const}.
\end{aligned}$$

Therefore, the optimum $A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y})$ which maximizes this part of $\mathcal{L}(\theta)$ is

$$A_t^*(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) = p(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) \propto p_t(\mathbf{X}_t) K^{\text{del}}(i|n_t).$$

which is the same form as given in Proposition 1.

B Training Objective

We estimate our objective $\mathcal{L}(\theta)$ by taking minibatches from the expectation $\mathcal{U}(t; 0, T) p_{0,t}(\mathbf{X}_0, \mathbf{X}_t, M_t) K^{\text{del}}(i|n_t) \delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})$. We first sample $t \sim \mathcal{U}(t; 0, T)$ and then take samples from our dataset $\mathbf{X}_0 \sim p_{\text{data}}(\mathbf{X}_0)$. In order to sample $p_{t|0}(\mathbf{X}_t, M_t|\mathbf{X}_0)$ we need to both add noise, delete dimensions and sample a mask variable. Since the Gaussian noising process is isotropic, we can add a suitable amount of noise to all dimensions of $\mathbf{X}_0$ and then delete dimensions of that noised full dimensional value. More specifically, we first sample $\tilde{\mathbf{X}}_t = (n_0, \tilde{\mathbf{x}}_t)$ with $\tilde{\mathbf{x}}_t \sim \mathcal{N}(\tilde{\mathbf{x}}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1 - \alpha_t) I_{n_0 d})$ for $\alpha_t = \exp\left(-\int_0^t \beta(s) \mathrm{d}s\right)$ using the analytic forward equations for the VP-SDE derived in [3]. Then we sample the number of dimensions to delete. This is simple to do when our rate function is independent of n except for the case when $n = 1$ at which it is zero. We simply sample a Poisson random variable with mean parameter $\int_0^t \overrightarrow{\lambda}_s \mathrm{d}s$ and then clamp its value such that the maximum number of possible components that are deleted is $n_0 - 1$. This gives the appropriate distribution over n, $p_{t|0}(n|n_0)$ as given in Section A.4. To sample which dimensions are deleted, we can sample $K^{\text{del}}(i_1|n_0) K^{\text{del}}(i_2|n_0 - 1) \dots K^{\text{del}}(i_{n_0 - n_t}|n_t + 1)$ from which we can create the mask M_t and apply it to $\tilde{\mathbf{X}}_t$ to obtain $\mathbf{X}_t$, $\mathbf{X}_t = M_t(\tilde{\mathbf{X}}_t)$. When $K^{\text{del}}(i|n) = 1/n$ this is especially simple to do by simply randomly permuting the components of $\tilde{\mathbf{X}}_t$, and then removing the final $n_0 - n_t$ components.

As is typically done in standard diffusion models, we parameterize s_t^θ in terms of a noise prediction network that predicts ϵ where $\mathbf{x}_t = \sqrt{\alpha_t} M_t(\mathbf{x}_0) + \sqrt{1 - \alpha_t}\epsilon$, $\epsilon \sim \mathcal{N}(0, I_{n_t d})$. We then re-weight the score loss in time such that we have a uniform weighting in time rather than the 'likelihood weighting' with g_t^2 [3, 21]. Our objective to learn s_t^θ then becomes

$$-\mathbb{E}_{\mathcal{U}(t;0,T) p_{\text{data}}(\mathbf{X}_0) p(M_t, n_t|\mathbf{X}_0) \mathcal{N}(\epsilon; 0, I_{n_t d})}\left[\|\epsilon_t^\theta(\mathbf{X}_t) - \epsilon\|^2\right]$$

with $\mathbf{x}_t = \sqrt{\alpha_t} M_t(\mathbf{x}_0) + \sqrt{1 - \alpha_t}\epsilon$, $s_t^\theta(\mathbf{X}_t) = \frac{-1}{\sqrt{1-\alpha_t}} \epsilon_t^\theta(\mathbf{X}_t)$.

Further, by using the parameterization given in Proposition 3, we can directly supervise the value of $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ by adding an extra term to our objective. We can treat the learning of $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ as a standard prediction task where we aim to predict n_0 given access to $\mathbf{X}_t$. A standard objective for learning $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ is then the cross entropy

$$\max_\theta \quad \mathbb{E}_{p_{0,t}(\mathbf{X}_0, \mathbf{X}_t)}\left[\log p_{0|t}^\theta(n_0|\mathbf{X}_t)\right]$$

Our augmented objective then becomes

$$\tilde{\mathcal{L}}(\theta) = T\mathbb{E}[-\frac{1}{2}\|\epsilon_t^\theta(\mathbf{X}_t)-\epsilon\|^2 - \overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) + \overrightarrow{\lambda}_t(n_t)\log \overleftarrow{\lambda}_t^\theta(\mathbf{Y}) + \overrightarrow{\lambda}_t(n_t)\log A_t^\theta(\mathbf{x}_t^{\text{add}}, i|\mathbf{Y}) + \gamma \log p_{0|t}^\theta(n_0|\mathbf{X}_t)] \tag{25}$$

where the expectation is taken with respect to

$$\mathcal{U}(t;0,T)p_{\text{data}}(\mathbf{X}_0)p(M_t,n_t|\mathbf{X}_0)\mathcal{N}(\epsilon;0,I_{n_t d})K^{\text{del}}(i|n_t)\delta_{\text{del}(\mathbf{X}_t,i)}(\mathbf{Y})$$

where $\mathbf{x}_t = \sqrt{\alpha_t}M_t(\mathbf{x}_0) + \sqrt{1-\alpha_t}\epsilon$ and γ is a loss weighting term for the cross entropy loss.

C Trans-Dimensional Diffusion Guidance

To guide an unconditionally trained model such that it generates datapoints consistent with conditioning information, we use the reconstruction guided sampling approach introduced in [9]. Our conditioning information will be the values for some of the components of $\mathbf{X}_0$, and thus the guidance should guide the generative process such that the rest of the components of the generated datapoint are consistent with those observed components. Following the notation of [9], we denote the observed components as $\mathbf{x}^a \in \mathbb{R}^{n_a d}$ and the components to be generated as $\mathbf{x}^b \in \mathbb{R}^{n_b d}$. Our trained score function $s_t^\theta(\mathbf{X}_t)$ approximates $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{X}_t)$ whereas we would like the score to approximate $\nabla_{\mathbf{x}_t}\log p_t(\mathbf{X}_t|\mathbf{x}_0^a)$. In order to do this, we will need to augment our unconditional score $s_t^\theta(\mathbf{X}_t)$ such that it incorporates the conditioning information.

We first focus on the dimensions of the score vector corresponding to $\mathbf{x}^a$. These can be calculated analytically from the forward process

$$\nabla_{\mathbf{x}_t^a}\log p(\mathbf{X}_t|\mathbf{x}_0^a) = \nabla_{\mathbf{x}_t^a}\log p_{t|0}(\mathbf{x}_t^a|\mathbf{x}_0^a, n_t)$$

with $p_{t|0}(\mathbf{x}_t^a|\mathbf{x}_0^a, n_t) = \mathcal{N}(\mathbf{x}_t^a; \sqrt{\alpha_t}\mathbf{x}_0^a, (1-\alpha_t)I_{n_a d})$. Note that we assume a correspondence between $\mathbf{x}_t^a$ and $\mathbf{x}_0^a$. For example, in video if we condition on the first and last frame, we assume that the first and last frame of the current noisy $\mathbf{x}_t$ correspond to $\mathbf{x}_0^a$ and guide them towards their observed values. For molecules, the point cloud is permutation invariant and so we can simply assume the first n_a components of $\mathbf{x}_t$ correspond to $\mathbf{x}_0^a$ and guide them to their observed values.

Now we analyse the dimensions of the score vector corresponding to $\mathbf{x}^b$. We split the score as

$$\nabla_{\mathbf{x}_t^b}\log p(\mathbf{X}_t|\mathbf{x}_0^a) = \nabla_{\mathbf{x}_t^b}\log p(\mathbf{x}_0^a|\mathbf{X}_t) + \nabla_{\mathbf{x}_t^b}\log p_t(\mathbf{X}_t)$$

$p(\mathbf{x}_0^a|\mathbf{X}_t)$ is intractable to calculate directly and so, following [9], we approximate it with $\mathcal{N}(\mathbf{x}_0^a; \hat{\mathbf{x}}_0^{\theta a}(\mathbf{X}_t), \frac{1-\alpha_t}{\alpha_t}I_{n_a d})$ where $\hat{\mathbf{x}}_0^{\theta a}(\mathbf{X}_t)$ is a point estimate of $\mathbf{x}_0^a$ given from $s_t^\theta(\mathbf{X}_t)$ calculated as

$$\hat{\mathbf{x}}_0^{\theta a}(\mathbf{X}_t) = \frac{\mathbf{x}_t^a + (1-\alpha_t)s_t^\theta(\mathbf{X}_t)^a}{\sqrt{\alpha_t}}$$

where again we have assumed a correspondence between $\mathbf{x}_t^a$ and $\mathbf{x}_0^a$. Our approximation for $\nabla_{\mathbf{x}_t^b}\log p(\mathbf{x}_0^a|\mathbf{X}_t)$ is then

$$\nabla_{\mathbf{x}_t^b}\log p(\mathbf{x}_0^a|\mathbf{X}_t) \approx -\nabla_{\mathbf{x}_t^b}\frac{\alpha_t}{2(1-\alpha_t)}\|\mathbf{x}_0^a - \hat{\mathbf{x}}_0^{\theta a}(\mathbf{X}_t)\|^2$$

which can be calculated by differentiating through the score network s_t^θ.

We approximate $\overleftarrow{\lambda}_t^*(\mathbf{X}_t|\mathbf{x}_0^a)$ and $A_t^*(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t, \mathbf{x}_0^a)$, with their unconditional forms $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ and $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$. We find this approximation still leads to valid generations because the guidance of the score network s_t^θ, results in $\mathbf{X}_t$ containing the conditioning information which in turn leads to $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ guiding the number of components in $\mathbf{X}_t$ to be consistent with the conditioning information too as verified in our experiments. Further, any errors in the approximation for $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ are fixed by further applications of the guided score function, highlighting the benefits of our combined autoregressive and diffusion based approach.

D Experiment Details

Our code is available at `https://github.com/andrew-cr/jump-diffusion`

D.1 Molecules

D.1.1 Network Architecture

Backbone For our backbone network architecture, we used the EGNN used in [8]. This is a specially designed graph neural network applied to the point cloud treating it as a fully connected graph. A special equivariant update is used, operating only on distances between atoms. We refer to [8] for the specific details on the architecture. We used the same size network as used in [8]'s QM9 experiments, specifically there are 9 layers, with a hidden node feature size of 256. The output of the EGNN is fed into a final output projection layer to give the score network output $s_t^\theta(\mathbf{X}_t)$.

Component number prediction To obtain $p_{0|t}^\theta(n_0|\mathbf{X}_t)$, we take the embedding produced by the EGNN before the final output embedding layer and pass it through 8 transformer layers each consisting of a self-attention block and an MLP block applied channel wise. Our transformer model dimension is 128 and so we project the EGNN embedding output down to 128 before entering into the transformer layers. We then take the output of the transformer and take the average embedding over all nodes. This embedding is then passed through a final projection layer to give softmax logits over the $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ distribution.

Autoregressive Distribution Our $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ network has to predict the position and features for a new atom when it is added to the molecule. Since the point cloud is permutation invariant, we do not need to predict i and so we just need to parameterize $A_t^\theta(\mathbf{y}^{\text{add}}|\mathbf{X}_t)$. We found the network to perform the best if the network first predicts the nearest atom to the new atom and then a vector from that atom to the location of the new atom. To achieve this, we first predict softmax logits for a distribution over the nearest atom by applying a projection to the embedding output from the previously described transformer block. During training, the output of this distribution can be directly supervised by a cross entropy loss. Given the nearest atom, we then need to predict the position and features of the new atom to add. We do this by passing in the embedding generated by the EGNN and original point cloud features into a new transformer block of the same size as that used for $p_{0|t}^\theta(n_0|\mathbf{X}_t)$. We also input the distances from the nearest atom to all other atoms in the molecule currently as an additional feature. To obtain the position of the new atom, we will take a weighted sum of all the vectors between the nearest atom and other atoms in the molecule. This is to make it easy for the network to create new atoms 'in plane' with existing atoms which is useful for e.g. completing rings that have to remain in the same plane. To calculate the weights for the vectors, we apply an output projection to the output of the transformer block. The new atom features (atom type and charge) are generated by a separate output projection from the transformer block. For the position and features, $A_t^\theta(\mathbf{y}^{\text{add}}|\mathbf{X}_t)$ outputs both a mean and a standard deviation for a Gaussian distribution. For the position distribution, we set the standard deviation to be isotropic to remain equivariant to rotations. In total our model has around 7.3 million parameters.

D.1.2 Training

We train our model for 1.3 million iterations at a batch size of 64. We use the Adam optimizer with learning rate 0.00003. We also keep a running exponential moving average of the network weights that is used during sampling as is standard for training diffusion models [2, 3, 16] with a decay parameter of 0.9999. We train on the 100K molecules contained in the QM9 training split. We model hydrogens explicitly. Training a model requires approximately 7 days on a single GPU which was done on an Academic cluster.

In [8] the atom type is encoded as a one-hot vector and diffused as a continuous variable along with the positions and charge values for all atoms. They found that multiplying the one-hot vectors by 0.25 to boost performance by allowing the atom-type to be decided later on in the diffusion process. We instead multiply the one-hot vectors by 4 so that atom-type is decided early on in the diffusion process which improves our guided performance when conditioning on certain atom-types being

present. We found our model is robust to this change and achieves similar sample quality to [8] as shown in Table 2.

When deleting dimensions, we first shuffle the ordering of the nodes and then delete the final $n_0 - n_t$ nodes. The cross entropy loss weighting in (25) is set to 1.

Following [8] we train our model to operate within the center of mass (CoM) zero subspace of possible molecule positions. The means, throughout the forward and backward process, the average position of an atom is 0. In our transdimensional framework, this is achieved by first deleting any atoms required under the forward component deletion process. We then move the molecule such that its CoM is 0. We then add CoM free noise such that the noisy molecule also has CoM$= 0$. Our score model s_t^θ is parameterized through a noise prediction model ϵ_t^θ which is trained to predict the CoM free noise that was added. Therefore, our score network learns suitable directions to maintain the process on the CoM$= 0$ subspace. For the position prediction from $A_t^\theta(\mathbf{y}^{\text{add}}|\mathbf{X}_t)$ we train it to predict the new atom position from the current molecules reference frame. When the new atom is added, we then update all atom positions such that CoM$= 0$ is maintained.

D.1.3 Sampling

During sampling we found that adding corrector steps [3] improved sample quality. Intuitively, corrector steps form a process that has $p_t(\mathbf{X})$ as its stationary distribution rather than the process progressing toward $p_0(\mathbf{X})$. We use the same method to determine the corrector step size ζ as in [3]. For the conditional generation tasks, we also found it useful to include corrector steps for the component generation process. As shown in [29], corrector steps in discrete spaces can be achieved by simulating with a rate that is the addition of the forward and backward rates. We achieve this in the context of trans-dimensional modeling by first simulating a possible insertion using $\overleftarrow{\lambda}_t^\theta$ and then simulating a possible deletion using $\overrightarrow{\lambda}_t$. We describe our overall sampling algorithm in Algorithm 2.

Algorithm 2: Sampling the Generative Process with Corrector Steps

Input: Number of corrector steps C
$t \leftarrow T$
$\mathbf{X} \sim p_{\text{ref}}(\mathbf{X}) = \mathbb{I}\{n = 1\}\mathcal{N}(\mathbf{x}; 0, I_d)$
while $t > 0$ **do**
  if $u < \overleftarrow{\lambda}_t^\theta(\mathbf{X})\delta t$ with $u \sim \mathcal{U}(0,1)$ **then**
    Sample $\mathbf{x}^{\text{add}}, i \sim A_t^\theta(\mathbf{x}^{\text{add}}, i|\mathbf{X})$
    $\mathbf{X} \leftarrow \text{ins}(\mathbf{X}, \mathbf{x}^{\text{add}}, i)$
  end
  $\mathbf{x} \leftarrow \mathbf{x} - \overleftarrow{\mathbf{b}}_t^\theta(\mathbf{X})\delta t + g_t\sqrt{\delta t}\epsilon$ with $\epsilon \sim \mathcal{N}(0, I_{nd})$
  for $c = [1, \ldots, C]$ **do**
    $\mathbf{x} \leftarrow \mathbf{x} + \zeta s_{t-\delta t}^\theta(\mathbf{X}) + \sqrt{2\zeta}\epsilon$ with $\epsilon \sim \mathcal{N}(0, I_{nd})$
    if $u < \overleftarrow{\lambda}_{t-\delta t}^\theta(\mathbf{X})\delta t$ with $u \sim \mathcal{U}(0,1)$ **then**
      Sample $\mathbf{x}^{\text{add}}, i \sim A_{t-\delta t}^\theta(\mathbf{x}^{\text{add}}, i|\mathbf{X})$
      $\mathbf{X} \leftarrow \text{ins}(\mathbf{X}, \mathbf{x}^{\text{add}}, i)$
    end
    if $u < \overrightarrow{\lambda}_{t-\delta t}(n)\delta t$ with $u \sim \mathcal{U}(0,1)$ **then**
      $\mathbf{X} \leftarrow \text{del}(\mathbf{X}, i)$ with $i \sim K^{\text{del}}(i|n)$
    end
  end
  $\mathbf{X} \leftarrow (n, \mathbf{x}), t \leftarrow t - \delta t$
end

D.1.4 Evaluation

Unconditional For our unconditional sampling evaluation, we start adding corrector steps when $t < 0.1T$ in the backward process and use 5 corrector steps without the corrector steps on the number of components. We set $\delta = 0.05$ for $t > 0.5T$ and $\delta = 0.001$ for $t < 0.5T$ such that the total number

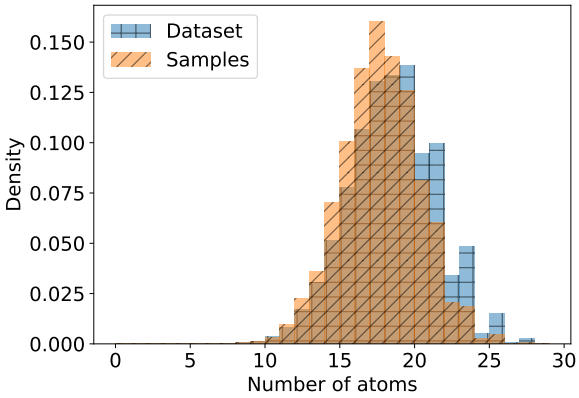


Figure 6: Distribution of the size of molecules in the QM9 dataset as measured through the number of atoms versus the distribution of the size of molecules generated by our unconditional model.

of network evaluations is 1000. We show the distribution of sizes of molecules generated by our model in Figure 6 and show more unconditional samples in Figure 7. We find our model consistently generates realistic molecules and achieves a size distribution similar to the training dataset even though this is not explicitly trained and arises from sampling our backward rate $\overleftarrow{\lambda}_t^\theta$. Since we are numerically integrating a continuous time process and approximating the true time reversal rate $\overleftarrow{\lambda}_t^*$, some approximation error is expected. For this experiment, sampling all of our models and ablations takes approximately 2 GPU days on Nvidia 1080Ti GPUs.

Conditional For evaluating applying conditional diffusion guidance to our model, we choose 10 conditioning tasks that each result in a different distribution of target dimensions. The task is to produce molecules that include at least a certain number of target atom types. We then guide the first set of atoms generated by the model to have these desired atom types. The tasks chosen are given in Table 5. Molecules in the training dataset that meet the conditions in each task have a different distribution of sizes. The tasks were chosen so that we have an approximately linearly increasing mean number of atoms for molecules that meet the condition. We also require that there are at least 100 examples of molecules that meet the condition within the training dataset.

For sampling when using conditional diffusion guidance, we use 3 corrector steps throughout the backward process with $\delta t = 0.001$. For these conditional tasks, we include the corrector steps on the number of components. We show the distribution of dimensions for each task from the training dataset and from our generated samples in Figure 8. Our metrics are calculated by first drawing 1000 samples for each conditioning task and then finding the Hellinger distance between the size distribution generated by our method (orange diagonal hashing in Figure 8) and the size distribution for molecules in the training dataset that match the conditions of the task (green no hashing in Figure 8). We find that indeed our model when guided by diffusion guidance can automatically produce a size distribution close to the ground truth size distribution found in the dataset for that conditioning value. We show samples generated by our conditionally guided model in Figure 9. We can see that our model can generate realistic molecules that include the required atom types and are of a suitable size. For this experiment, sampling all of our models and ablations takes approximately 13 GPU days on Nvidia 1080Ti GPUs.

Interpolations For our interpolations experiments, we follow the set up of [8] who train a new model conditioned on the polarizability of molecules in the dataset. We train a conditional version of our model which can be achieved by simply adding in the polarizability as an additional feature input to our backbone network and re-using all the same hyperparameters. We show more examples of interpolations in Figure 10.

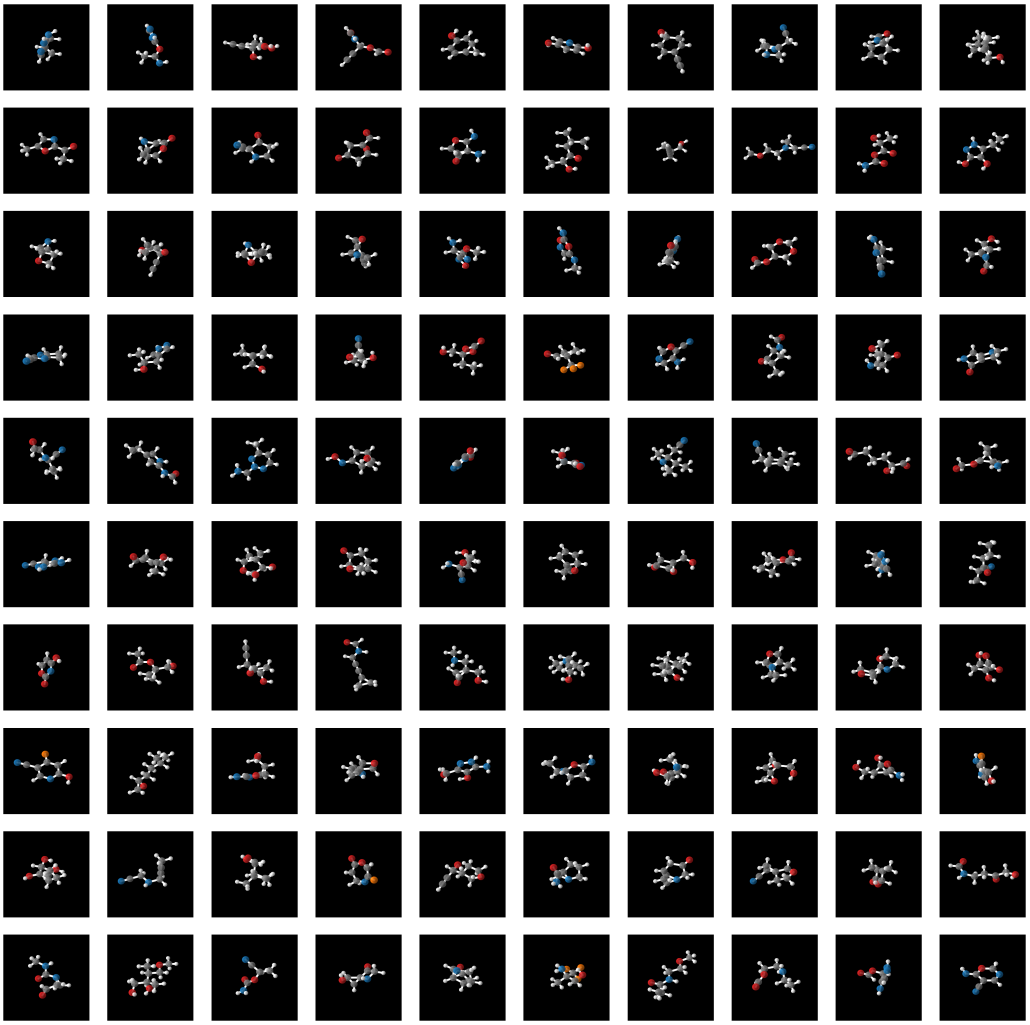


Figure 7: Unconditional samples from our model.

D.1.5 Ablations

For our main model, we set $\overrightarrow{\lambda}_{t<0.1T} = 0$ to ensure that all dimensions are added with enough generation time remaining for the diffusion process to finalize all state values. To verify this setting, we compare its performance with $\overrightarrow{\lambda}_{t<0.03T} = 0$ and $\overrightarrow{\lambda}_{t<0.3T} = 0$. We show our results in Table 6. We find that the $\overrightarrow{\lambda}_{t<0.03T} = 0$ setting to generate reasonable sample quality but incur some extra dimension error due to the generative process sometimes observing a lack of dimensions near $t = 0$ and adding too many dimensions. We observed the same effect in the paper for when setting $\overrightarrow{\lambda}_t$ to be constant for all t in Table 3. Further, the setting $\overrightarrow{\lambda}_{t<0.3T} = 0$ also results in increased dimension error due to there being less opportunity for the guidance model to supervise the number of dimensions. We find that $\overrightarrow{\lambda}_{t<0.1T} = 0$ to be a reasonable trade-off between these effects.

D.1.6 Uniqueness and Novelty Metrics

We here investigate sample diversity and novelty of our unconditional generative models. We measure uniqueness by computing the chemical graph corresponding to each generated sample and measure what proportion of the 10000 produced samples have a unique chemical graph amongst this set of 10000 as is done in [8]. We show our results in Table 7 and find our TDDM method to have slightly lower levels of uniqueness when compared to the fixed dimension diffusion model baseline.

Table 5: The 10 conditioning tasks used for evaluation. The number of each atom type required for the task is given in columns $2-5$ whilst the average number of atoms in molecules that meet this condition in the training dataset is given in the 6th column.

Task	Carbon	Nitrogen	Oxygen	Fluorine	Mean Number of Atoms
1	4	1	2	1	11.9
2	4	3	1	1	13.0
3	5	2	1	1	13.9
4	6	0	1	1	14.6
5	5	3	1	0	16.0
6	6	3	0	0	17.2
7	6	1	2	0	17.7
8	7	1	1	0	19.1
9	8	1	0	0	19.9
10	8	0	1	0	21.0

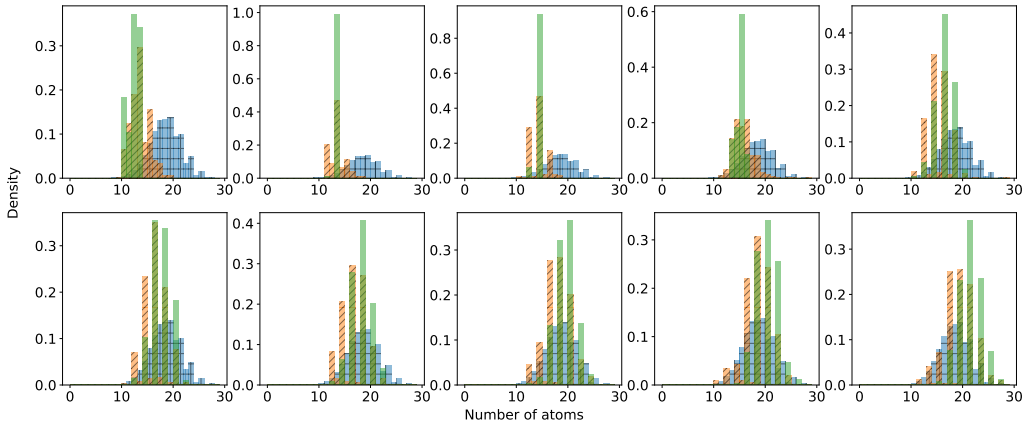


Figure 8: Distribution of molecule sizes for each conditioning task. Tasks $1-5$ are shown left to right in the top row and tasks $6-10$ are shown left to right in the bottom row. We show the unconditional size distribution from the dataset in blue vertical/horizontal hashing, the size distribution of our conditionally generated samples in orange diagonal hashing and finally the size distribution for molecules in the training dataset that match the conditions of each task (the ground truth size distribution) in green no hashing.

Measuring novelty on generative models trained on the QM9 dataset is challenging because the QM9 dataset contains an exhaustive enumeration of all molecules that satisfy certain predefined constraints [46], [8]. Therefore, if a novel molecule is produced it means the generative model has failed to capture some of the physical properties of the dataset and indeed it is found in [8] that during training, as the model improved, novelty decreased. Novelty is therefore not typically included in evaluating molecular diffusion models. For completeness, we include the novelty scores in Table 7 as a comparison to the results presented in [8] Appendix C. We find that our samples are closer to the statistics of the training dataset whilst still producing 'novel' samples at a consistent rate.

D.2 Video

D.2.1 Dataset

We used the VP^2 benchmark, which consists of 35 000 videos, each 35 frames long. The videos are evenly divided among seven tasks, namely: `push {red, green, blue} button, open {slide, drawer}, push {upright block, flat block} off table`. The 5000 videos for each task were collected using a scripted task-specific policy operating in the RoboDesk environment [40]. They sample an action vector at every step during data generation by adding i.i.d. Gaussian

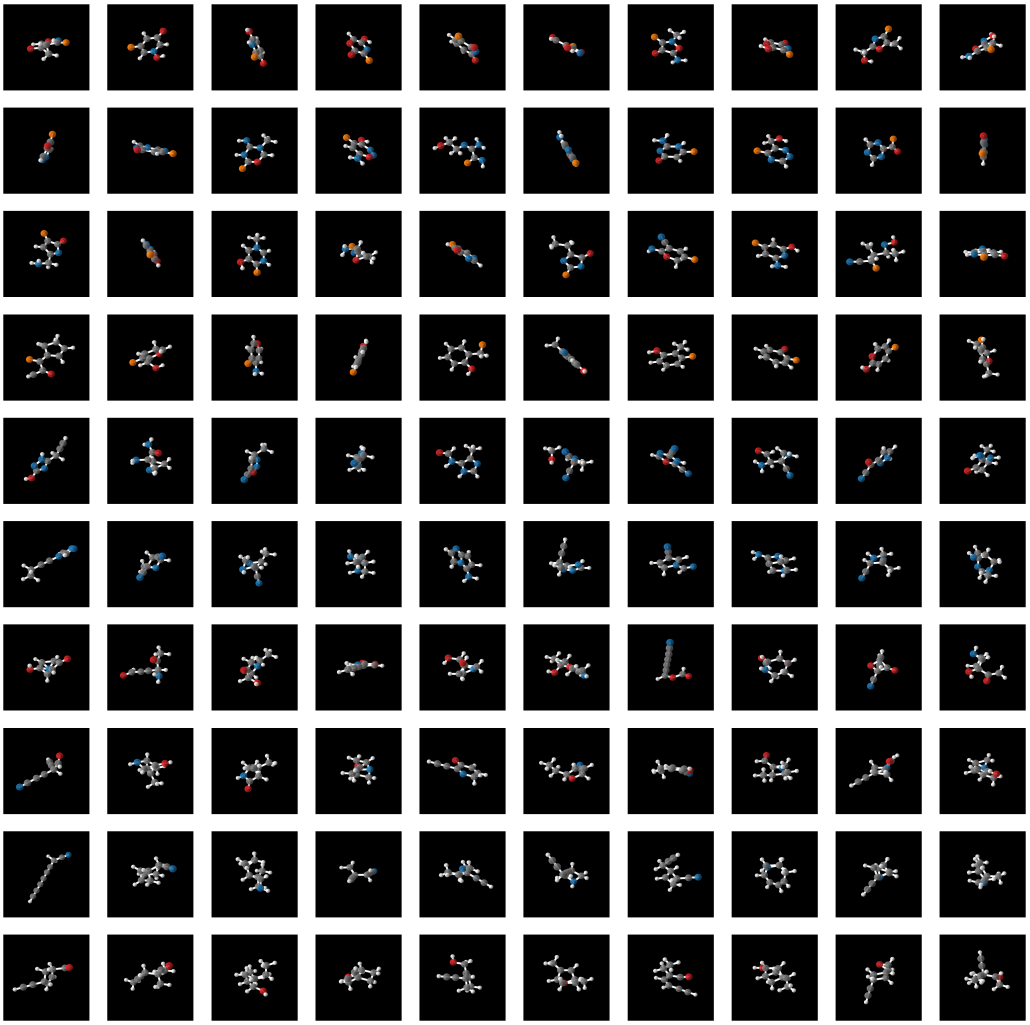


Figure 9: Samples generated by our model when conditional diffusion guidance is applied. Each row represents one task with task 1 at the top, down to task 10 at the bottom. For each task, 10 samples are shown in each row.

noise to each dimension of the action vector output by the scripted policy. For each task, they sample 2500 videos with noise standard deviation 0.1 and 2500 videos with standard deviation 0.2. We filter out the lower-quality trajectories sampled with noise standard deviation 0.2, and so use only the 17 500 videos (2500) per task with noise standard deviation 0.1. We convert these videos to 32×32 resolution and then, so that the data we train on has varying lengths, we create each training example by sampling a length l from a uniform distribution over $\{2, \ldots, 35\}$ and then taking a random l-frame subset of the video.

D.2.2 Forward Process

The video domain differs from molecules in two important ways. The first is that videos cannot be reasonably treated as a permutation-invariant set. This is because the order of the frames matters. Secondly, generating a full new component for the molecules with a single pass autoregressive network is feasible, however, a component for the videos is a full frame which is challenging for a single pass autoregressive network to generate. We design our forward process to overcome these challenges.

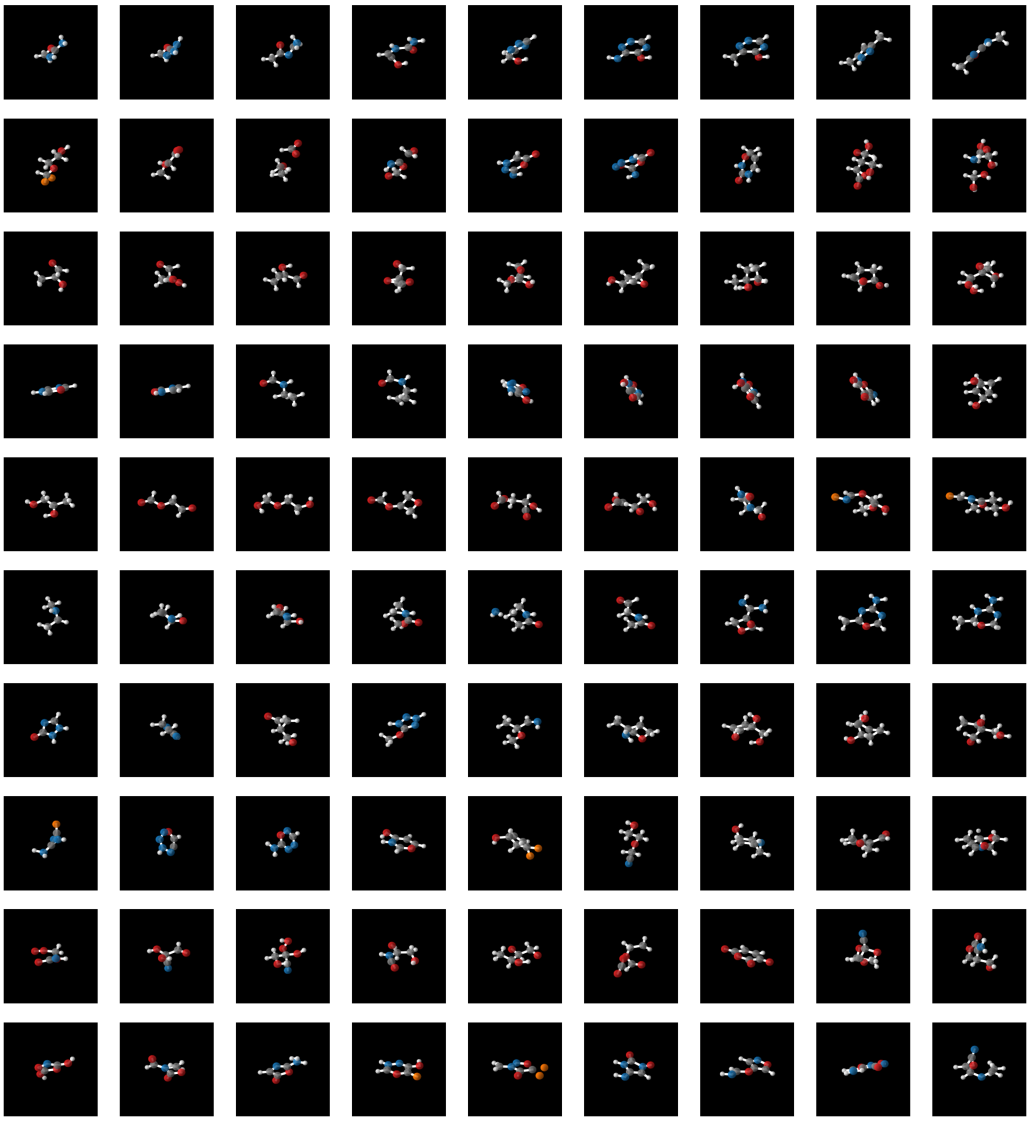


Figure 10: Interpolations showing a sequence of generations for linearly increasing polarizability from 39 Bohr^3 to 66 Bohr^3 with fixed random noise. Each row shows an individual interpolation with Bohr^3 increasing from left to right.

Table 6: Ablation of when to set the forward rate to 0 on the conditional molecule generation task. We report dimension error as the average Hellinger distance between the generated and ground truth conditional dimension distributions as well as average sample quality metrics. Metrics are reported after 620k training iterations.

Method	Dimension Error	% Atom stable	% Molecule Stable	% Valid
$\overrightarrow{\lambda}_{t<0.03T} = 0$	$0.227_{\pm 0.16}$	$91.5_{\pm 3.7}$	$56.5_{\pm 9.8}$	$72.0_{\pm 11}$
$\overrightarrow{\lambda}_{t<0.1T} = 0$	$0.162_{\pm 0.071}$	$92.4_{\pm 2.8}$	$53.9_{\pm 12}$	$72.7_{\pm 9.6}$
$\overrightarrow{\lambda}_{t<0.3T} = 0$	$0.266_{\pm 0.11}$	$92.0_{\pm 3.2}$	$53.5_{\pm 13}$	$66.6_{\pm 12}$

Table 7: Uniqueness and novelty metrics on unconditional molecule generation. We produce 10000 samples for each method and measure validity using RDKit. Uniquenss is judged as whether the chemical graph is unique amongst the 10000 produced samples. Amongst the valid and unique molecules, we then find the percentage that have a chemical graph not present in the training dataset.

Method	% Valid	% Valid and Unique	Percentage of Valid and Unique Molecules that are Novel
FDDM [8]	91.9	90.7	65.7
TDDM (ours)	92.3	89.9	53.6
TDDM, const $\overrightarrow{\lambda}_t$	86.7	84.4	56.9
TDDM, $\overrightarrow{\lambda}_{t<0.9T} = 0$	89.4	86.1	51.3
TDDM w/o Prop. 3	87.1	85.9	63.3

We define our forward process to delete frames in a random order. This means that during generation, frames can be generated in any order in the reverse process, enabling more conditioning tasks since we can always ensure that whichever frames we want to condition on are added first. Further, we use a non-isotropic noise schedule by adding noise just to the frame that is about to be deleted. Once it is deleted, we then start noising the next randomly chosen frame. This is so that, in the backward direction, when a new frame is added, it is simply Gaussian noise. Then the score network will fully denoise that new frame before the next new frame is added. We now specify exactly how our forward process is constructed.

We enable random-order deletion by applying an initial shuffling operation occurring at time $t = 0$. Before this operation, we represent the video $\mathbf{x}$ as an ordered sequence of frames, $\mathbf{x}_0 = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n_0}]$. During shuffling, we sample a random permutation π of the integers $1, \dots, n_0$. Then the frames are kept in the same order, but annotated with an index variable so that we have $\mathbf{x}_{0^+} = [(\mathbf{x}_{0^+}^{(1)}, \pi(1)), (\mathbf{x}_{0^+}^{(2)}, \pi(2)), \dots, (\mathbf{x}_{0^+}^{(n_0)}, \pi(n_0))]$.

We will run the forward process from $t = 0$ to $t = 100N$. We will set the forward rate such we delete down from n_t to $n_t - 1$ at time $(N - n_t + 1)100$. This is achieved heuristically by setting

$$\overrightarrow{\lambda}_t(n_t) = \begin{cases} 0 & \text{for } t < (N - n_t + 1)100, \\ \infty & \text{for } t \geq (N - n_t + 1)100. \end{cases}$$

We can see that at time $t = (N - n_t + 1)100$ we will quickly delete down from n_t to $n_t - 1$ at which point $\overrightarrow{\lambda}_t(n_t)$ will become 0 thus stopping deletion until the process arrives at the next multiple of 100 in time. When we hit a deletion event, we delete the frame from $\mathbf{X}_t$ that has the current highest index variable $\pi(n)$. In other words

$$K^{\text{del}}(i|\mathbf{X}_t) = \begin{cases} 1 & \text{for } n_t = \mathbf{x}_t^{(i)}[2], \\ 0 & \text{otherwise} \end{cases}$$

where we use $\mathbf{x}_t^{(i)}[2]$ to refer to the shuffle index variable for the ith current frame in $\mathbf{x}_t$.

We now provide an example progression of the forward deletion process. Assume we have $n_0 = 4$, $N = 5$ and sample a permutation such that $\pi(1) = 3, \pi(2) = 2, \pi(3) = 4$, and $\pi(4) = 1$. Initially

the state is augmented to include the shuffle index. Then the forward process progresses from $t = 0$ to $t = 500$ with components being deleted in descending order of the shuffle index

$$\begin{aligned}
\mathbf{x}_{0^+} &= [(\mathbf{x}_t^{(1)}, 3), (\mathbf{x}_t^{(2)}, 2), (\mathbf{x}_t^{(3)}, 4), (\mathbf{x}_t^{(4)}, 1)] \\
\mathbf{x}_{100^+} &= [(\mathbf{x}_t^{(1)}, 3), (\mathbf{x}_t^{(2)}, 2), (\mathbf{x}_t^{(3)}, 4), (\mathbf{x}_t^{(4)}, 1)] \\
\mathbf{x}_{200^+} &= [(\mathbf{x}_t^{(1)}, 3), (\mathbf{x}_t^{(2)}, 2), (\mathbf{x}_t^{(4)}, 1)] \\
\mathbf{x}_{300^+} &= [(\mathbf{x}_t^{(2)}, 2), (\mathbf{x}_t^{(4)}, 1)] \\
\mathbf{x}_{400^+} &= [(\mathbf{x}_t^{(4)}, 1)]
\end{aligned}$$

In this example, due to the random permutation sampled, the final video frame remained after all others had been deleted. Note that the order of frames is preserved as we delete frames in the forward process although the spacing between them can change as we delete frames in the middle.

Between jumps, we use a noising process to add noise to frames. The noising process is non-isotropic in that it adds noise to different frames at different rates such that the a frame is noised only in the time window immediately preceding its deletion. For component $i \in [1, \ldots, n_t]$, we set the forward noising process such that $p_{t|0}(\mathbf{x}_t^{(i)}|\mathbf{x}_0^{(i)}, M_t) = \mathcal{N}(\mathbf{x}_t^{(i)}; \mathbf{x}_0^{(i)}, \sigma_t(\mathbf{x}_t^{(i)})^2)$ where $\mathbf{x}_0^{(i)}$ is the clean frame corresponding to $\mathbf{x}_t^{(i)}$ as given by the mask M_t and $\sigma_t(\mathbf{x}_t^{(i)})$ follows

$$\sigma_t(\mathbf{x}_t^{(i)}) = \begin{cases} 0 & \text{for } t < (N - \mathbf{x}_t^{(i)}[2])100, \\ 100 & \text{for } t > (N - \mathbf{x}_t^{(i)}[2])100, \\ t - (N - \mathbf{x}_t^{(i)}[2])100 & \text{for } (N - \mathbf{x}_t^{(i)}[2])100 \leq t \leq (N - \mathbf{x}_t^{(i)}[2] + 1)100 \end{cases}$$

where we again use $\mathbf{x}_t^{(i)}[2]$ for the shuffle index of component i. This is the VE-SDE from [3] applied to each frame in turn. We note that we only add noise to the state values on not the shuffle index itself. The SDE parameters that result in the VE-SDE are $\overrightarrow{\mathbf{b}}_t = 0$ and $\overrightarrow{g}_t = \sqrt{2t - 2(N - \mathbf{x}_t^{(i)}[2])100}$.

D.2.3 Sampling the Backward Process

When t is not at a multiple of 100, the forward process is purely adding Gaussian noise, and so the reverse process is also purely operating on the continuous dimensions. We use the Heun sampler proposed by [16] to update the continuous dimensions in this case, and also a variation of their discretisation of t - specifically to update from e.g. $t = 600$ to $t = 500$, we use their discretization of t as if the maximum value was 100 and then offset all values by 500.

To invert the dimension deletion process, we can use Proposition 3 to derive our reverse dimension generation process. We re-write our parameterized $\overleftarrow{\lambda}_t^\theta$ using Proposition 3 as

$$\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) = \overrightarrow{\lambda}_t(n_t + 1)\mathbb{E}_{p_{0|t}^\theta(n_0|\mathbf{X}_t)}\left[\frac{p_{t|0}(n_t + 1|n_0)}{p_{t|0}(n_t|n_0)}\right]$$

At each time multiple of 100 in the backward process, we will have an opportunity to add a component. At this time point, we estimate the expectation with a single sample $n_0 \sim p_{0|t}^\theta(n_0|\mathbf{X}_t)$. If $n_0 > n_t$ then $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) = \infty$. The new component will then be added at which point $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ becomes 0 for the remainder of this block of time due to n_t becoming $n_t + 1$. If $n_0 = n_t$ then $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t) = 0$ and no new component is added. $\overleftarrow{\lambda}_t^\theta(\mathbf{X}_t)$ will continue to be 0 for the remainder of the backward process once an opportunity to add a component is not used.

When a new frame is added, we use $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ to decide where the frame is added and its initial value. Since when we delete a frame it is fully noised, $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ can simply predict Gaussian noise for the new frame $\mathbf{y}^{\text{add}}$. However, $A_t^\theta(\mathbf{y}^{\text{add}}, i|\mathbf{X}_t)$ will still learn to predict a suitable location i to place the new frame such that backward process is the reversal of the forward.

We give an example simulation from the backward generative process in Figure 11.

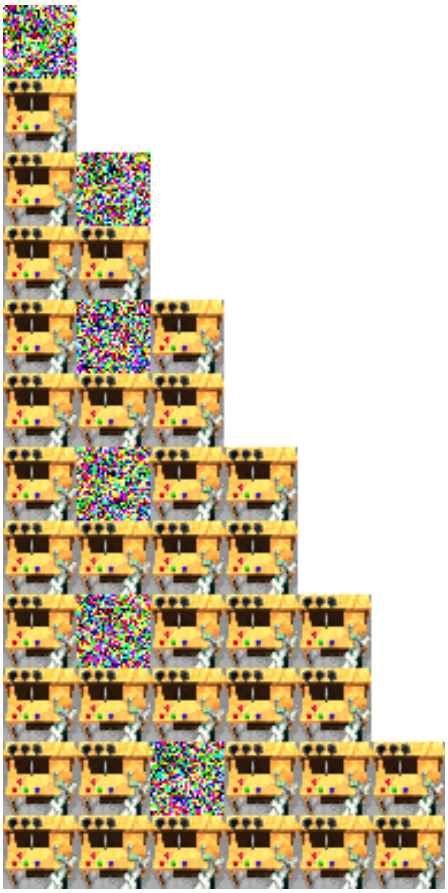


Figure 11: An example simulation of the backward generative process conditioned on the first and last frame. Note how the process first adds a new frame and then fully denoises it before adding the next frame. Since the first and last frame are very similar, the process produces a short video.

D.2.4 Network Architecture

Our video diffusion network architecture is based on the U-net used by [42], which takes as input the index of each frame within the video, and uses the differences between these indices to control the interactions between frames via an attention mechanism. Since, during generation, we do not know the final position of each frame within the $\mathbf{x}_0$, we instead pass in its position within the ordered sequence $\mathbf{x}_t$.

One further difference is that, since we are perform non-isotropic diffusion, the standard deviation of the added noise will differ between frames. We adapt to this by performing preconditioning, and inputting the timestep embedding, separately for each frame $\mathbf{x}_t^{(i)}$ based on $\sigma_t(\mathbf{x}_t^{(i)})$ instead of basing them on the global diffusion timestep t. Our timestep embedding and pre- and post-conditioning of network inputs/outputs are as suggested by [16], other than being done on a per-frame basis. The architecture from [42] with these changes applied then gives us our score network s_t^θ.

While it would be possible to train a single network that estimates the score and all quantities needed for modelling jumps, we chose to train two separate networks in order to factorize our exploration of the design space. These were the score network s_t^θ, and the rate and index prediction network modeling $p_{0|t}^\theta(n_0|\mathbf{X}_t)$ and $A_t^\theta(i|\mathbf{X}_t)$. The rate and index prediction network is similar to the first half of the score network, in that it uses all U-net blocks up to and including the middle one. We then flatten the $512 \times 4 \times 4$ hidden state for each frame after this block such that, for an n_t frame input, we obtain a $n_t \times 8192$ hidden state. These are fed through a 1D convolution with kernel size 2 and zero-padding of size 1 on each end, reducing the hidden state to $(n_t + 1) \times 128$, which is in turn fed

through a ReLU activation function. This hidden state is then fed into three separate heads. One head maps it to the parameters of $A_t^\theta(i|\mathbf{X}_t)$ via a 1D convolution of kernel size 3. The output of size $(n_t + 1)$ is fed through a softmax to provide the categorical distribution $A_t^\theta(i|\mathbf{X}_t)$. The second head averages the hidden state over the "frame" dimension, producing a 128-dimensional vector. This is fed through a single linear layer and a softmax to parameterize $p^\theta_{0|t}(n_0|\mathbf{X}_t)$. Finally, the third head consists of a 1D convolution of kernel size 3 with 35 output channels. The $(n_t + 1) \times 35$ output is fed through a softmax to parameterize distributions over the number of frames that were deleted from $\mathbf{X}_0$ which came before the first in $\mathbf{x}_t$, the number of frames from $\mathbf{X}_0$ which were deleted between each pair of frames in $\mathbf{x}_t$, and the number deleted after the last frame in $\mathbf{x}_t$. We do not use this head at inference-time but found that including it improved the performance of the other heads by helping the network learn better representations.

For a final performance improvement, we note that under our forward process there is only ever one "noised" frame in $\mathbf{x}_t$, while there are sometimes many clean frames. Since the cost of running our architecture scales with the number of frames, running it on many clean frames may significantly increase the cost while providing little improvement to performance. We therefore only feed into the architecture the "noised" frame, the two closest "clean" frames before it, and the two closest "clean" frames after it. See our released source code for the full implementation of this architecture.

D.2.5 Training

To sample t during training, we adapt the log-normal distribution suggested by [16] in the context of isotropic diffusion over a single image. To apply it to our non-isotropic video diffusion, we first sample which frames have been deleted, which exist with no noise, and which have had noise added, by sampling the timestep from a uniform distribution and simulating our proposed forward process. We then simply change the noise standard deviation for the noisy frame, replacing it with a sample from the log-normal distribution. The normal distribution underlying our log-normal has mean -0.6 and standard deviation 1.8. This can be interpreted as sampling the timestep from a mixture of log-normal distributions, $\frac{1}{N}\sum_{i=0}^{N-1} \mathcal{LN}(t - 100i; -0.6, 1.8^2)$. Here, the mixture index i can be interpreted as controlling the number of deleted frames.

We use the same loss weighting as [16] but, similarly to our use of preconditioning, compute the weighting separately for each frame $\mathbf{x}_t^{(i)}$ as a function of $\sigma_t(\mathbf{x}_t^{(i)})$ to account for the non-isotropic noise.

D.2.6 Perceptual Quality Metrics

We now verify that our reverse process does not have any degradation in quality during the generation as more dimensions are added. We generate 10000 videos and throw away the 278 that were sampled to have only two frames. We then compute the FID score for individual frames in each of the remaining 9722 videos. We group together the scores for all the first frames to be generated in the reverse process and then for the second frame to be generated and so on. We show our results in Table 8. We find that when a frame is inserted has no apparent effect on perceptual quality and conclude that there is no overall degradation in quality as our sampling process progresses. We note that the absolute value of these FID scores may not be meaningful due to the RoboDesk dataset being far out of distribution for the Inception network used to calculate FID scores. We can visually confirm good sample quality from Figure 5.

Table 8: FID for video frames grouped by when they were inserted during sampling.

1st	2nd	3rd	3rd last	2nd last	last
34.2	34.9	34.7	34.2	34.1	34.4

E Broader Impacts

In this work, we presented a general method for performing generative modeling on datasets of varying dimensionality. We have not focused on applications and instead present a generic method.

Along with other generic methods for generative modeling, we must consider the potential negative social impacts that these models can cause when inappropriately used. As generative modeling capabilities increase, it becomes simpler to generate fake content which can be used to spread misinformation. In addition to this, generative models are becoming embedded into larger systems that then have real effects on society. There will be biases present within the generations created by the model which in turn can reinforce these biases when the model's outputs are used within wider systems. In order to mitigate these harms, applications of generative models to real world problems must be accompanied with studies into their biases and potential ways they can be misused. Further, public releases of models must be accompanied with model cards [47] explaining the biases, limitations and intended uses of the model.

4.1 Additional Notes

We note that as part of this work, we applied continuous diffusion models to the task of generating molecular point clouds. Point cloud data has a natural permutation invariance as no matter the ordering in which the nodes are represented when stacked into a vector $\mathbb{R}^{N\times 3}$, these all refer to the same underlying molecule. When applying diffusion models to this data, we treated the position data as standard vectors for simplicity. However, this disregards the permutation symmetry present in the data. For example, when training the denoising model, the fixed vector representation, in effect, applies a completely uninformed matching between the predicted node positions and true node positions. In concurrent work, Klein et al. 2023 experiment with adding a permutation matching step to the calculation of the loss in order to take the permutation invariance structure into account during training. They find this results in shorter sampling paths during generation that can be useful for simulation efficiency. It would be an exciting avenue for further work to combine this matching approach with our jump-diffusion model.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Trans-Dimensional Generative Modeling via Jump Diffusion Models
Publication Status	Published
Publication Details	Campbell, A., Harvey, W., Weilbach, C., De Bortoli, V., Rainforth, T. and Doucet, A., 2024. Trans-dimensional generative modeling via jump diffusion models. *Advances in Neural Information Processing Systems*, *36*.

Student Confirmation

Student Name:	Andrew Campbell		
Contribution to the Paper	- Lead author - Jointly conceptualized the study with collaborators - Derived the initial mathematical framework with refinements from collaborators - Wrote the code for and ran all of the experiments on small molecules - Wrote the majority of the manuscript with additions and refinements from collaborators. - Conceptualized and proved Propositions 2, 3 with verification from collaborators. Wrote intuitive version of Proposition 1 proof.		
Signature		Date	08-Jan-2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Arnaud Doucet, Senior Staff Research Scientist Google DeepMind			
Supervisor comments I agree with the description of the contributions.			
Signature		Date	09-Jan-2025

This completed form should be included in the thesis, at the end of the relevant chapter.

5

Improved Motif-Scaffolding with SE(3) Flow Matching

Improved motif-scaffolding with SE(3) flow matching

Jason Yim *jyim@csail.mit.edu*
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Andrew Campbell *campbell@stats.ox.ac.uk*
Department of Statistics
University of Oxford

Emile Mathieu *ebm32@cam.ac.uk*
Department of Engineering
University of Cambridge

Andrew Y. K. Foong *andrewfoong@microsoft.com*
Microsoft Research AI4Science

Michael Gastegger *mgastegger@microsoft.com*
Microsoft Research AI4Science

José Jiménez-Luna *jjimenezluna@microsoft.com*
Microsoft Research AI4Science

Sarah Lewis *sarahlewis@microsoft.com*
Microsoft Research AI4Science

Victor Garcia Satorras *victorgar@microsoft.com*
Microsoft Research AI4Science

Bastiaan S. Veeling *basveeling@microsoft.com*
Microsoft Research AI4Science

Frank Noé *franknoe@microsoft.com*
Microsoft Research AI4Science

Regina Barzilay *regina@csail.mit.edu*
Computer Science and Articial Intelligence Laboratory
Massachusetts Institute of Technology

Tommi S. Jaakkola *tommi@csail.mit.edu*
Computer Science and Articial Intelligence Laboratory
Massachusetts Institute of Technology

Reviewed on OpenReview: *https://openreview.net/forum?id=fa1ne8xDGn*

Abstract

Protein design often begins with the knowledge of a desired function from a motif which motif-scaffolding aims to construct a functional protein around. Recently, generative models have achieved breakthrough success in designing scaffolds for a range of motifs. However, generated scaffolds tend to lack structural diversity, which can hinder success in wet-lab validation. In this work, we extend FrameFlow, an SE(3) flow matching model for protein backbone generation, to perform motif-scaffolding with two complementary approaches. The first is *motif amortization*, in which FrameFlow is trained with the motif as input using a data augmentation strategy. The second is *motif guidance*, which performs scaffolding using an estimate of the conditional score from FrameFlow without additional training. On a benchmark of 24 biologically meaningful motifs, we show our method achieves 2.5 times more designable and unique motif-scaffolds compared to state-of-the-art. Code: `https://github.com/microsoft/protein-frame-flow`

1 Introduction

A common task in protein design is to create proteins with functional properties conferred through a pre-specified arrangement of residues known as a *motif*. The problem is to design the remainder of the protein, called the *scaffold*, that harbors the motif. Motif-scaffolding is widely used, with applications to vaccine and enzyme design (Procko et al., 2014; Correia et al., 2014; Jiang et al., 2008; Siegel et al., 2010). For this problem, diffusion models have greatly advanced capabilities in designing new scaffolds (Wu et al., 2023; Trippe et al., 2022; Ingraham et al., 2023). While experimental wet-lab validation is the ultimate test for evaluating a scaffold, in this work we focus on improving performance under computational validation of scaffolds following prior works. *In-silico* success is defined as satisfying the designability[1] criteria which has been found to correlate well with wet-lab success (Wang et al., 2021). The current state-of-the-art, RFdiffusion (Watson et al., 2023), fine-tunes a pre-trained RosettaFold (Baek et al., 2023) neural network with SE(3) diffusion (Yim et al., 2023b) and is able to successfully scaffold the majority of motifs in a recent benchmark.[2] However, RFdiffusion suffers from low scaffold diversity which can hinder chances of a successful design. Moreover, the large model size and pre-training used in RFdiffusion makes it slow to train and difficult to deploy on smaller machines. In this work, we present a lightweight and easy-to-train model with improved performance.

Our method adapts an existing SE(3) flow matching model, FrameFlow (Yim et al., 2023a), for motif-scaffolding. We develop two approaches: (i) *motif amortization*, and (ii) *motif guidance* as illustrated in Fig. 1. Motif amortization simply trains a *conditional* model with the motif as additional input when generating the scaffold. We use data augmentation to amortize over all possible motifs in our training set and aid in generalization to new motifs. Motif guidance relies on a Bayesian approach, using an *unconditional* FrameFlow model to sample the scaffold residues, while the motif residues are guided at each step to their final desired positions. An unconditional model in this context is one that generates the full protein backbone without distinguishing between the motif and scaffold. Motif guidance was described in Wu et al. (2023) for SE(3) diffusion. In this work, we develop the extension to SE(3) flow matching.

The two approaches differ in whether to use an conditional model or to re-purpose an unconditional model for conditional generation. Motif guidance has the advantage that any unconditional model can be used to readily perform motif scaffolding without the need for additional task-specific training. To provide a controlled comparison, we train unconditional and conditional versions of FrameFlow on a dataset of monomers from the Protein Data Bank (PDB) (Berman et al., 2000). Our results provide a clear comparison of the modeling choices made when performing motif-scaffolding with FrameFlow. We find that FrameFlow with

[1] A metric based on using ProteinMPNN (Dauparas et al., 2022) and AlphaFold2 (Jumper et al., 2021) to determine the quality of a protein backbone.

[2] First introduced in RFdiffusion as a benchmark of 24 single-chain motifs successfully solved across prior works published.

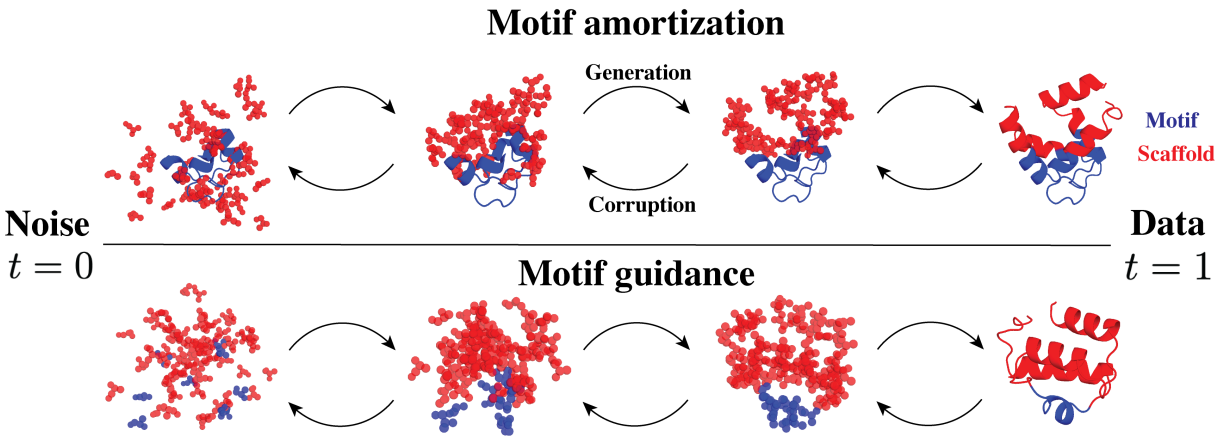


Figure 1: We present two strategies for motif-scaffolding. **Top**: motif amortization trains a flow model to condition on the motif (blue) and generate the scaffold (red). During training, only the scaffold is corrupted with noise. **Bottom**: motif guidance re-purposes a flow model that is trained to generate the full protein for motif-scaffolding. During generation, the motif residues are guided to reconstruct the true motif at $t = 1$ while the flow model will adjust the scaffold trajectory to be consistent with the motif.

both motif amortization and guidance surpasses the performance of RFdiffusion, as measured by the number of structurally *unique* scaffolds[3] that pass the designability criterion.

This work is structured as follows. Sec. 2 provides background on SE(3) flow matching. We present our main contribution extending FrameFlow for motif-scaffolding in Sec. 3. We develop motif amortization for flow matching while motif guidance, originally developed for diffusion models, follows after drawing connections between flow matching and diffusion models. Next we discuss related works Sec. 4 and present empirical results Sec. 5. Our contributions are the following:

- We extend FrameFlow with two fundamentally different approaches for motif-scaffolding: motif amortization and motif guidance. We are the first to extend conditional generation techniques with SE(3) flow matching and apply them to motif-scaffolding. With all other settings kept constant, we perform a empirical study of how each approach performs.
- On a benchmark of biologically meaningful motifs, we show our method can successfully scaffold 20 out of 24 motifs in the motif-scaffolding benchmark which is equivalent to previous state-of-the-art, while achieving 2.5 times more unique, designable scaffolds. Our results demonstrate the importance of measuring diversity to detect mode collapse.

2 Background

Flow matching (FM) (Lipman et al., 2023; Albergo et al., 2023) is a simulation-free method for training continuous normalizing flows (CNFs) (Chen et al., 2018). CNFs are deep generative models that generates data by integrating an ordinary differential equation (ODE) over a learned vector field. Recently, flow matching has been extended to Riemannian manifolds (Chen & Lipman, 2023), which we rely on to model protein backbones via the local frame SE(3) representation. Sec. 2.1 gives an introduction to Riemannian flow matching. Sec. 2.2 then describes how SE(3) flow matching is applied to protein backbones.

2.1 Flow matching on Riemannian manifolds

On a manifold $\mathcal{M}$, a CNF $\phi_t(\cdot) : \mathcal{M} \to \mathcal{M}$ is defined via an ODE along a time-dependent vector field $v(z, t) : \mathcal{M} \times \mathbb{R} \to \mathcal{T}_z\mathcal{M}$ where $\mathcal{T}_z\mathcal{M}$ is the tangent space of the manifold at $z \in \mathcal{M}$ and time is $t \in [0, 1]$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi_t(z_0) = v(\phi_t(z_0), t), \;\; \phi_0(z_0) = z_0. \tag{1}$$

[3]The number of unique scaffolds is defined as the number of structural clusters. See Sec. 5.1

Starting with $z_0 \sim p_0$ from an easy-to-sample prior distribution p_0, simulating samples according to Eq. (1) induces a new distribution referred as the push-forward $p_t = [\phi_t]_* p_0$. One wishes to find a vector field v such that the push-forward $p_{t=1} = [\phi_{t=1}]_* p_0$ (at $t = 1$) matches the data distribution p_1. Such a vector field v is in general not available in closed-form, but can be learned by regressing conditional vector fields $u(z_t, t|z_1) = \frac{\mathrm{d}}{\mathrm{d}t} z_t$ where $z_t = \phi_t(z_0|z_1)$ interpolates between endpoints $z_0 \sim p_0$ and $z_1 \sim p_1$. A natural choice for z_t is the geodesic path: $z_t = \exp_{z_0}\left(t \log_{z_0}(z_1)\right)$, where $\exp_{z_0}$ and $\log_{z_0}$ are the exponential and logarithmic maps at the point z_0. The conditional vector field takes the following form: $u(z_t, t|z_1) = \log_{z_t}(z_1)/(1-t)$. The key insight of conditional[4] flow matching (CFM) (Lipman et al., 2023) is that training a neural network $\hat{v}$ to regress the conditional vector field u is equivalent to learning the unconditional vector field v. This corresponds to minimizing

$$\mathcal{L} = \mathbb{E}_{\mathcal{U}(t;[0,1]), p_1(z_1), p_0(z_0)} \left[\| u(z_t, t|z_1) - \hat{v}(z_t, t) \|_g^2 \right] \tag{2}$$

where $\mathcal{U}(t;[0,1])$ is the uniform distribution for $t \in [0,1]$ and $\|\cdot\|_g^2$ is the norm induced by the Riemannian metric $g : \mathcal{TM} \times \mathcal{TM} \to \mathbb{R}$. Samples can then be generated by integrating the ODE in Eq. (1) with Euler steps using the learned vector field $\hat{v}$ in place of v.

2.2 Generative modeling on protein backbones

The atom positions of each residue in a protein backbone can be parameterized by an element $T \in \mathrm{SE}(3)$ of the special Euclidean group $\mathrm{SE}(3)$ (Jumper et al., 2021; Yim et al., 2023b). We refer to $T = (r, x)$ as a (local) frame consisting of a rotation $r \in \mathrm{SO}(3)$ and translation vector $x \in \mathbb{R}^3$. The protein backbone is made of N residues, meaning it can be parameterized by N frames denoted as $\mathbf{T} = [T^{(1)}, \ldots, T^{(N)}] \in \mathrm{SE}(3)^N$. We use bold face to refer to vectors of all the residues, superscripts to refer to residue indices, and subscripts refer to time. Details of the $\mathrm{SE}(3)^N$ backbone parameterization can be found in App. B.1.

We use $\mathrm{SE}(3)$ flow matching to parameterize a generative model over the $\mathrm{SE}(3)^N$ representation of protein backbones. The application of Riemannian flow matching to $\mathrm{SE}(3)$ was previously developed in Yim et al. (2023a); Bose et al. (2023). Endowing $\mathrm{SE}(3)$ with the product left-invariant metric, the $\mathrm{SE}(3)$ manifold effectively behaves as the product manifold $\mathrm{SE}(3) = \mathrm{SO}(3) \times \mathbb{R}^3$ (App. D.3 of Yim et al. (2023b)). The vector field over $\mathrm{SE}(3)$ can then be decomposed as $v^{(n)}_{\mathrm{SE}(3)}(\cdot, t) = (v^{(n)}_{\mathbb{R}}(\cdot, t), v^{(n)}_{\mathrm{SO}(3)}(\cdot, t))$. Our goal is train a neural network to parameterize the learned vector fields,

$$\hat{v}^{(n)}_{\mathbb{R}}(\mathbf{T}_t, t) = \frac{\hat{x}^{(n)}_1(\mathbf{T}_t) - x^{(n)}_t}{1-t}, \qquad \hat{v}^{(n)}_{\mathrm{SO}(3)}(\mathbf{T}_t, t) = \frac{\log_{r^{(n)}_t}(\hat{r}^{(n)}_1(\mathbf{T}_t))}{1-t}. \tag{3}$$

The outputs of the neural network are *denoised* predictions $\hat{x}^{(n)}_1$ and $\hat{r}^{(n)}_1$ which are used to calculate the vector fields in Eq. (3). The loss becomes

$$\mathcal{L}_{\mathrm{SE}(3)} = \mathbb{E}\left[\left\| \mathbf{u}_{\mathrm{SE}(3)}(\mathbf{T}_t, t|\mathbf{T}_1) - \hat{\mathbf{v}}_{\mathrm{SE}(3)}(\mathbf{T}_t, t) \right\|^2_{\mathrm{SE}(3)} \right] \tag{4}$$

$$= \mathbb{E}\Big[\| \mathbf{u}_{\mathbb{R}}(\mathbf{x}_t, t|\mathbf{x}_1) - \hat{\mathbf{v}}_{\mathbb{R}}(\mathbf{T}_t, t) \|^2_{\mathbb{R}} + \left\| \mathbf{u}_{\mathrm{SO}(3)}(\mathbf{r}_t, t|\mathbf{r}_1) - \hat{\mathbf{v}}_{\mathrm{SO}(3)}(\mathbf{T}_t, t) \right\|^2_{\mathrm{SO}(3)} \Big] \tag{5}$$

where the expectation is taken over $\mathcal{U}(t;[0,1])$, $p_1(\mathbf{T}_1)$, $p_0(\mathbf{T}_0)$. We have used bold-face for collections of elements, i.e. $\hat{\mathbf{v}}(\cdot) = [\hat{v}^{(1)}(\cdot), \ldots, \hat{v}^{(N)}(\cdot)]$. Our prior is chosen as $p_0(\mathbf{T}_0) = \mathcal{U}(\mathrm{SO}(3))^N \otimes \overline{\mathcal{N}}(0, I_3)^N$, where $\mathcal{U}(\mathrm{SO}(3))$ is the uniform distribution over $\mathrm{SO}(3)$ and $\overline{\mathcal{N}}(0, I_3)$ is the isotropic Gaussian where samples are centered to the origin. Details of $\mathrm{SE}(3)$ flow matching such as architecture and hyperparameters closely follow FrameFlow (Yim et al., 2023a), details of which are provided in App. B.2.

3 Motif-scaffolding with FrameFlow

We describe our two strategies for performing motif-scaffolding with the FrameFlow model: motif amortization (Sec. 3.1) and motif guidance (Sec. 3.2). Recall the full protein backbone is given by $\mathbf{T} =$

[4]Unfortunately the meaning of "conditional" is overloaded. The conditionals will be clear from the context.

Figure 2: **Motif data augmentation.** Each protein in the dataset does not come with pre-defined motif-scaffold annotations. Instead, we construct plausible motifs at random to simulate sampling from the distribution of motifs and scaffolds.

$\{T^{(1)}, T^{(2)}, \ldots, T^{(N)}\} \in \mathrm{SE}(3)^N$. The residues can be separated into the motif $\mathbf{T}^M = \{T^{(i_1)}, \ldots, T^{(i_k)}\}$ of length k where $\{i_1, \ldots, i_k\} \subset \{1, \ldots, N\}$ are motif residue indices, and the scaffold $\mathbf{T}^S$ is all the remaining residues, such that $\mathbf{T} = \mathbf{T}^M \cup \mathbf{T}^S$. The task can then be framed as the problem of sampling from the conditional distribution $p(\mathbf{T}^S|\mathbf{T}^M)$.

3.1 Motif amortization

We train a variant of FrameFlow that additionally takes the motif as input when generating scaffolds (and keeping the motif fixed). Formally, we model a motif-conditioned CNF via the following ODE,

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi_t(\mathbf{T}_0^S|\mathbf{T}^M) = v(\phi_t, t|\mathbf{T}^M), \quad \phi_0(\mathbf{T}_0^S|\mathbf{T}^M) = \mathbf{T}_0^S. \tag{6}$$

The flow ϕ_t transforms a prior density over scaffolds along time, inducing a density $p_t(\cdot|\mathbf{T}^M) = [\phi_t]_* p_0(\cdot|\mathbf{T}^M)$. We use the same prior as in Sec. 2.2: $p_0(\mathbf{T}_0^S|\mathbf{T}^M) = p_0(\mathbf{T}_0^S)$. FrameFlow is trained to predict the conditional vector field $u(\mathbf{T}_t^S, t|\mathbf{T}_1^S, \mathbf{T}^M)$ where $\mathbf{T}_t^S$ is defined by interpolating along the geodesic path, $\mathbf{T}_t^S = \exp_{\mathbf{T}_0^S}\left(t \log_{\mathbf{T}_0^S}(\mathbf{T}_1^S)\right)$. The implication is that u is conditionally independent of the motif $\mathbf{T}^M$ given $\mathbf{T}_1^S$. This simplifies our formulation to $u(\mathbf{T}_t^S, t|\mathbf{T}_1^S, \mathbf{T}^M) = u(\mathbf{T}_t^S, t|\mathbf{T}_1^S)$ that is defined in Sec. 2.2. However, when we learn the vector field, the model needs to condition on $\mathbf{T}^M$ since the motif placement $\mathbf{T}^M$ contains information on the true scaffold positions $\mathbf{T}_1^S$. The training loss becomes,

$$\mathbb{E}\left[\left\|\mathbf{u}_{\mathrm{SE}(3)}(\mathbf{T}_t^S, t|\mathbf{T}_1^S) - \hat{\mathbf{v}}_{\mathrm{SE}(3)}(\mathbf{T}_t^S, t|\mathbf{T}^M)\right\|^2_{\mathrm{SE}(3)}\right] \tag{7}$$

where the expectation is taken over $\mathcal{U}(t; [0, 1])$, $p(\mathbf{T}^M)$, $p_1(\mathbf{T}_1^S|\mathbf{T}^M)$, $p_0(\mathbf{T}_0^S)$. The above expectation requires access to the motif and scaffold distributions, $p(\mathbf{T}^M)$ and $p_1(\mathbf{T}_1^S|\mathbf{T}^M)$, during training. Future work can look into incorporating known motif-scaffolds such as the CDR loops on antibodies (Dunbar et al., 2014). While some labels exist for which residues correspond to the functional motif, the vast majority of protein structures in the PDB do not have labels. We instead utilize unlabeled PDB structures to perform data augmentation (see Sec. 3.1.1) that allows sampling a wide range of motifs and scaffolds.

To learn the motif-conditioned vector field $\hat{v}_t$, we use the FrameFlow architecture with a 1D mask as additional input with a 1 at the location of the motif and 0 elsewhere. To maintain SE(3)-equivariance, we zero-center the motif and initial noise sample from $p_0(\mathbf{T}_0^S|\mathbf{T}^M)$. Zero-centering the motif also prevents the model from using the motif offset from the origin to memorize scaffold locations which helps generalization.

3.1.1 Data augmentation.

The flow matching loss from Eq. (7) involves sampling from $p(\mathbf{T}^M)$ and $p_1(\mathbf{T}_1^S|\mathbf{T}^M)$, which we do not have access to, but can be approximated using unlabeled structures from the PDB. Our pseudo-labeled motifs and scaffolds are generated as follows (also depicted in Fig. 2). First, a protein structure is sampled from the PDB dataset. Second, a random number of residues are selected to be the starting locations of each motif. Third, additional residues are appended onto each motif thereby extending their lengths. The length of each motif is randomly sampled such that the total number of motif residues is between $\gamma_{\min}$ and $\gamma_{\max}$ percent of all the residues. We use $\gamma_{\min} = 0.05$ and $\gamma_{\max} = 0.5$ to ensure at least a few residues are used as the motif

but not more than half the protein. Finally, the remaining residues are treated as the scaffold and corrupted. The motif and scaffold are treated as samples from $p(\mathbf{T}^M)$ and $p_1(\mathbf{T}_1^S|\mathbf{T}^M)$ respectively. Importantly, each protein will be re-used on subsequent epochs where new motifs and scaffolds will be sampled. Our pseudo motif-scaffolds cover a wide range of scenarios that cover multiple motifs of different lengths.

The lack of functional annotations in the PDB requires training over all possible motif-scaffold annotations to handle new scenarios our method may encounter in real world scenarios. In our experiments, we evaluate how this data augmentation strategy transfers to real motif-scaffolding tasks. A similar strategy is used in image infilling where image based diffusion models are trained to infill randomly masked crops of images to approximate real image infilling scenarios (Saharia et al., 2022). Motif-scaffolding data augmentation was mentioned in RFdiffusion but without algorithmic detail. Since RFdiffusion does not release training code, we implemented our own data augmentation algorithm in Algorithm 1.

3.2 Motif guidance

We now present an alternative Bayesian approach to motif-scaffolding that does not involve learning a motif-conditioned flow model. As such it does not require having access to motifs at training time, but only at sampling time. This can be useful when an unconditional generative flow model is already available at hand and additional training is too costly. The idea behind motif guidance, first described as a special case of TDS (Wu et al., 2023) using diffusion models, is to use the desired motif $\mathbf{T}^M$ to bias the model's generative trajectory such that the motif residues end up in their known positions. The scaffold residues follow a trajectory that create a consistent whole protein backbone, thus achieving motif-scaffolding.

The key insight comes from connecting flow matching to diffusion models to which motif guidance can be applied. The following ODE describes the relationship between the vector field $\hat{\mathbf{v}}$ in flow models – learned by minimizing CFM objective in Eq. (5) – and the Stein score $\nabla \log p_t(\mathbf{T}_t)$,

$$\mathrm{d}\mathbf{T}_t = \hat{\mathbf{v}}(\mathbf{T}_t, t)\mathrm{d}t = \left[f(\mathbf{T}_t, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(\mathbf{T}_t) \right] \mathrm{d}t. \tag{8}$$

The gradient is taken with respect to the backbone at time t which we omit for brevity, i.e. $\nabla = \nabla_{\mathbf{T}_t}$. Eq. (8) shows the ODE used to sample from flow models can be written as the probability flow ODE used in diffusion models (Song et al., 2020) with f and g as the drift and diffusion coefficients. The derivation of Eq. (8) requires standard linear algebra and calculus for our choice of vector field (see App. D).

Our goal is to sample from the conditional $p(\mathbf{T}|\mathbf{T}^M)$ from which we can extract $p(\mathbf{T}^S|\mathbf{T}^M)$. The benefit of Eq. (8) is we can manipulate the score term to achieve this goal. We modify the above to be conditioned on the motif $\mathbf{T}^M$ followed by an application of Bayes rule where $\nabla \log p_t(\mathbf{T}_t|\mathbf{T}^M) = \nabla \log p_t(\mathbf{T}_t) + \nabla \log p_t(\mathbf{T}^M|\mathbf{T}_t)$.

$$\begin{aligned}
\mathrm{d}\mathbf{T}_t &= \left[f(\mathbf{T}_t, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(\mathbf{T}_t|\mathbf{T}^M) \right] \mathrm{d}t && (9) \\
&= \left[f(\mathbf{T}_t, t) - \frac{1}{2} g(t)^2 \Big(\nabla \log p_t(\mathbf{T}_t) + \nabla \log p_t(\mathbf{T}^M|\mathbf{T}_t) \Big) \right] \mathrm{d}t \\
&= \Big[\underbrace{\hat{\mathbf{v}}_{\mathrm{SE}(3)}(\mathbf{T}_t, t)}_{\text{unconditional pred.}} - \tfrac{1}{2} g(t)^2 \underbrace{\nabla \log p_t(\mathbf{T}^M|\mathbf{T}_t)}_{\text{guidance term}} \Big] \mathrm{d}t. && (10)
\end{aligned}$$

We can interpret Eq. (9) as doing unconditional generation by following $\hat{\mathbf{v}}_{\mathrm{SE}(3)}(\mathbf{T}, t)$ while $\nabla \log p_t(\mathbf{T}^M|\mathbf{T}_t)$ guides the noised residues so as to be consistent with the true motif. Doob's H-transform ensures Eq. (9) will sample from $p(\mathbf{T}|\mathbf{T}^M)$ (Didi et al., 2023). The conditional score $\nabla \log p_t(\mathbf{T}^M|\mathbf{T}_t)$ is unknown, yet it can be approximated by marginalising out $\mathbf{T}_1$ and using the neural network's denoised output (Song et al., 2022; Chung et al., 2022; Wu et al., 2023),

$$\begin{aligned}
p_t(\mathbf{T}^M|\mathbf{T}_t) &= \int p(\mathbf{T}^M|\mathbf{T}_1) p_{1|t}(\mathbf{T}_1|\mathbf{T}_t) \mathrm{d}\mathbf{T}_1 && (11) \\
&\approx \int p(\mathbf{T}^M|\mathbf{T}_1) \delta_{\hat{\mathbf{T}}_1^M(\mathbf{T}_t)}(\mathbf{T}_t) \mathrm{d}\mathbf{T}_1 = p(\mathbf{T}^M|\hat{\mathbf{T}}_1^M(\mathbf{T}_t)). && (12)
\end{aligned}$$

We now have the choice to define the likelihood in Eq. (12) to have higher probability the closer it is to the desired motif:

$$p(\mathbf{T}^M|\hat{\mathbf{T}}_1^M(\mathbf{T}_t)) \propto \exp\left(-\|\mathbf{x}^M - \hat{\mathbf{x}}_1^M(\mathbf{T}_t)\|_{\mathbb{R}}^2/\omega_t^2\right) \exp\left(-\|\mathbf{r}^M - \hat{\mathbf{r}}_1^M(\mathbf{T}_t)\|_{\mathrm{SO}(3)}^2/\omega_t^2\right), \tag{13}$$

which is inversely proportional to the distance from the desired motif. Following SE(3) flow matching, Eq. (9) becomes factorized into the translation and rotation components. Plugging $p(\mathbf{T}^M|\hat{\mathbf{T}}_1^M(\mathbf{T}_t))$ in Eq. (9), we arrive at the following ODE we may sample $p(\mathbf{T}|\mathbf{T}^M)$ from

$$\text{Translations: } \mathrm{d}\mathbf{x}_t = \left[\hat{\mathbf{v}}_{\mathbb{R}}(\mathbf{T}_t, t) + \tfrac{1}{2}g(t)^2\nabla_{\mathbf{x}_t}\|\mathbf{x}^M - \hat{\mathbf{x}}_1^M(\mathbf{T}_t)\|_{\mathbb{R}}^2/\omega_t^2\right]\mathrm{d}t. \tag{14}$$

$$\text{Rotations: } \mathrm{d}\mathbf{r}_t = \left[\hat{\mathbf{v}}_{\mathrm{SO}(3)}(\mathbf{T}_t, t) + \tfrac{1}{2}g(t)^2\nabla_{\mathbf{r}_t}\|\mathbf{r}^M - \hat{\mathbf{r}}_1^M(\mathbf{T}_t)\|_{\mathrm{SO}(3)}^2/\omega_t^2\right]\mathrm{d}t. \tag{15}$$

ω_t is a hyperparameter that controls the magnitude of the guidance towards the desired motif which we set to $\omega_t^2 = (1-t)^2/(t^2 + (1-t)^2)$ as done in Pokle et al. (2023); Song et al. (2021). While different choices of $g(t)$ are possible, Pokle et al. (2023) proposed to use $g(t) = (1-t)/t$ with the motivation that this matches the diffusion coefficient for the diffusion SDE that matches the marginals of the flow ODE. For completeness, we provide the proof for $g(t)$ in App. D. A similar calculation is non-trivial for SO(3), hence we use the same $g(t)$ as a reasonnable heuristic and observe good performance as done in (Wu et al., 2023).

4 Related work

Conditional diffusion and flows. The development of conditional generation methods for diffusion and flow models is an active area of research. Two popular diffusion techniques that have been extended to flow matching are classifier-free guidance (CFG) (Dao et al., 2023; Ho & Salimans, 2022; Zheng et al., 2023) and reconstruction guidance (Pokle et al., 2023; Ho et al., 2022; Song et al., 2022; Chung et al., 2022). Motif guidance is an application of reconstruction guidance for motif-scaffolding. Motif amortization is most related to data-dependent couplings (Albergo et al., 2023), where a flow is learned with conditioning of partial data.

Motif-scaffolding. Wang et al. (2021) first formulated motif-scaffolding using deep learning. SMCDiff (Trippe et al., 2022) was the first proposed diffusion model for motif-scaffolding using Sequential Monte Carlo (SMC). Twisted Diffusion Sampler (TDS) (Wu et al., 2023) later improved upon SMCDiff using reconstruction guidance for each particle in SMC. Our motif guidance method follows from TDS (with one particle) by deriving the equivalent guidance vector field from its conditional score counterpart. RFdiffusion (Watson et al., 2023) fine-tunes a pre-trained neural network with motif-conditioned diffusion training. Our FrameFlow-amortization approach in principle follows RFdiffusion's diffusion training, but differs in (i) using flow matching, (ii) not relying expensive pre-training, and (iii) uses a 3× smaller neural network[5]. Didi et al. (2023) provides a survey of structure-based motif-scaffolding methods while proposing Doob's h-transform for motifs-scaffolding. EvoDiff (Alamdari et al., 2023) differs in using a sequence-based diffusion model that performs motif-scaffolding with language model-style masked generation but performance falls short of RFdiffusion and TDS.

5 Experiments

In this section, we report the results of training FrameFlow for motif-scaffolding. Sec. 5.1 describes training, sampling, and metrics. Our main results on motif-scafolding are reported in Sec. 5.2 on the benchmark introduced in RFdiffusion. Additional motif-scaffolding analysis is provided in App. G.

5.1 Set-up

Training. We train two FrameFlow models. FrameFlow-amortization is trained with motif amortization as described in Sec. 3.1 with data augmentation using hyperparameters: $\gamma_{\min} = 0.05$ so the motif is never

[5]FrameFlow uses 16.8 million parameters compared to RFdiffusion's 59.8 million.

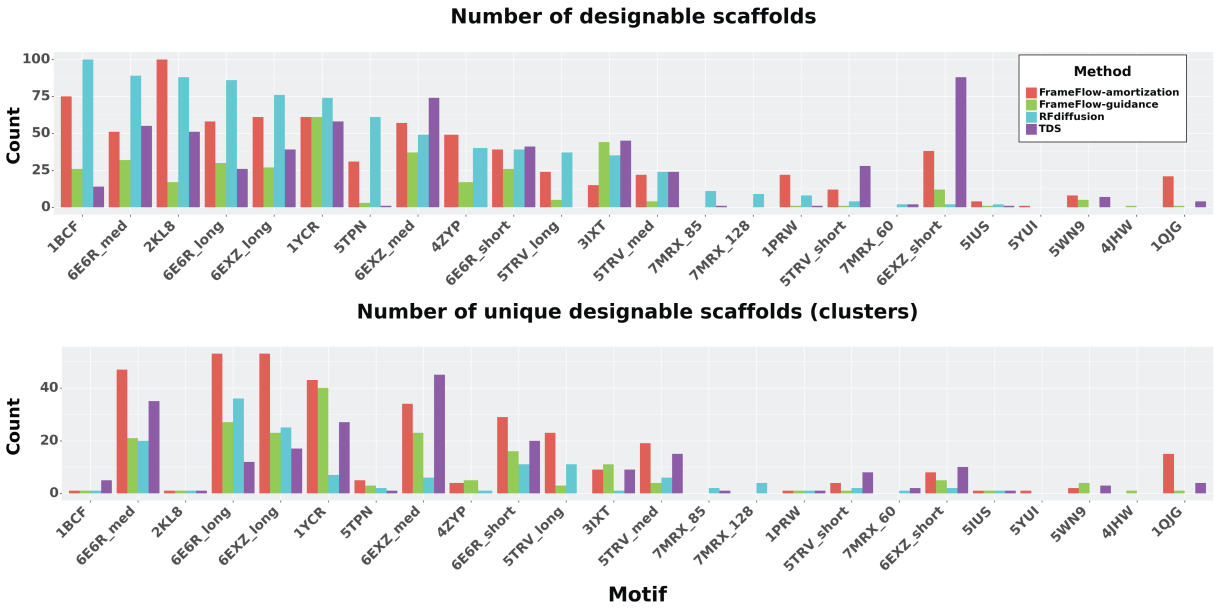


Figure 3: **Motif-scaffolding results.** Top plot: RFdiffusion achieves the most designable scaffolds amongst all methods in 9/24 test motifs compared to FrameFlow-amortization's 7/24 and TDS' 6/24; 2/24 are ties. Bottom plot: However, we observe that RFdiffusion produces the highest number of unique designable scaffolds for only 2 out of the 24 test motifs. Therefore, previous approaches that only measure designability (top plot) may be misleading since those generative models that may have the best designability can also be repeatedly sampling similar scaffolds. This demonstrates the need to measure diversity alongside designability and use the number of unique designable scaffolds as the metric of success.

degenerately small and $\gamma_{\max} = 0.5$ to avoid motif being the majority of the backbone. FrameFlow-guidance, to be used in motif guidance, is trained unconditionally on full backbones. Since unconditional generation is not our focus, we leave the unconditional performance to App. F where we see the performance is slightly worse than RFdiffusion – as we will see, the motif-scaffolding performance is better. Both models are trained using the filtered PDB monomer dataset introduced in FrameDiff. We use the ADAM optimizer (Kingma & Ba, 2014) with learning rate 0.0001. We train each model for 6 days on 2 A6000 NVIDIA GPUs with dynamic batch sizes depending on the length of the proteins in each batch — a technique from FrameDiff.

Sampling. We use the Euler-Maruyama integrator with 500 timesteps for all sampling. Following the motif-scaffolding benchmark proposed in RFdiffusion, we sample 100 scaffolds for each of the 24 monomer motifs[6]. For each motif, the method must sample novel scaffolds with different lengths and different motif locations along the sequence. The benchmark measures how well a method can generalize beyond the native scaffolds for a set of biologically important motifs.

Hyperparameters. Our hyperparameters for neural network architecture, optimizer, and sampling steps all follow the best settings found in FrameFlow (Yim et al., 2023a). We leave hyperparameter search as a future work since it is not the focus of this work.

5.2 Motif-scaffolding results

Baselines. We consider RFdiffusion and the Twisted Diffusion Sampler (TDS) as baselines. RFdiffusion's performance is reported based on their published samples. TDS reported motif-scaffolding results with arbitrary scaffold lengths that deviated the benchmark. Therefore, we re-ran TDS with their best settings using $k = 8$ particles on the RFdiffusion benchmark. We refer to **FrameFlow-amortization** as our results with motif amortization while **FrameFlow-guidance** uses motif guidance.

Metrics. Previously, motif-scaffolding was only evaluated through samples passing *designability* (**Des.**). For a description of designability see App. E. Within the set of designable scaffolds, we also calculate the

[6]The benchmark has 25 motifs, but the motif 6VW1 involves multiple chains that FrameFlow cannot handle.

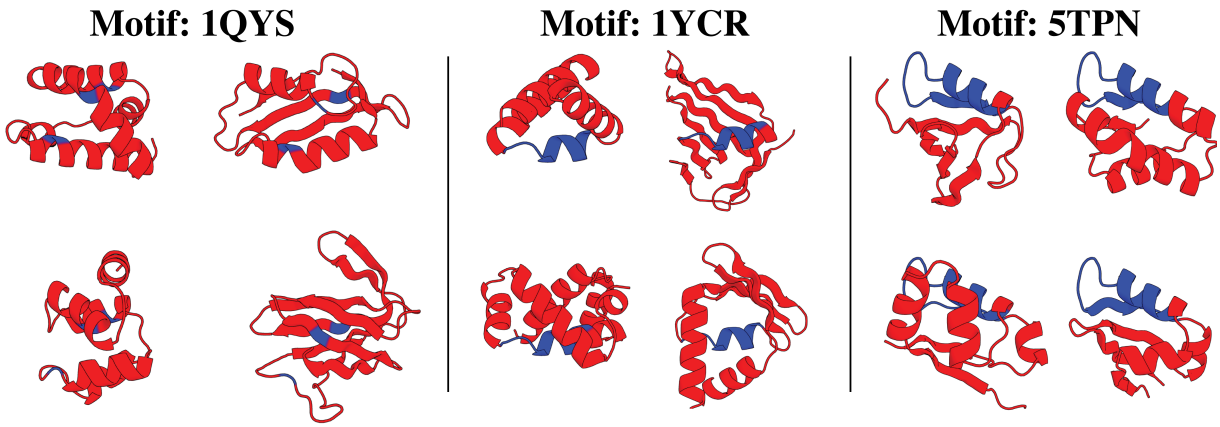


Figure 4: **FrameFlow-amortization diversity.** In blue is the motif while red is the scaffold. For each motif (1QJG, 1YCR, 5TPN), we show FrameFlow-amortization can generate scaffolds of different lengths and various secondary structure elements for the same motif. Each scaffold is in a unique cluster to showcase the samples'structural diversity.

diversity (**Div.**) as the number of structurally unique clusters. This is crucial since designability can be manipulated to have a 100% success rate by always sampling the same scaffold with trivial changes. In real world scenarios, diversity is desired to gain the most informative feedback from expensive wet-lab experiments (Yang et al., 2019). Thus diversity provides an additional data point to check for mode collapse where the model is sampling the same scaffold repeatedly. Clusters are computed using MaxCluster (Herbert & Sternberg, 2008) with TM-score threshold set to 0.5.

Benchmark. Fig. 3 shows how each method fares against each other in designability and diversity on each motif of the motif-scaffolding benchmark. While it appears RFdiffusion gets lots of successful scaffolds, the number of *unique* scaffolds is far lower than both our FrameFlow approaches. TDS achieves lower designable scaffolds on average, but demonstrates strong performance on a small subset of motifs. There are some motifs that only RFdiffusion can solve (7MRX_85, 7MRX_128) while FrameFlow is able to solve cases RFdiffusion cannot (1QJG, 4JHW, 5YUI).

Table 1: Motif-scaffolding aggregate metrics

Method	Solved (↑)	Div. (↑)	Speed (↓)
FrameFlow-amort.	**20**	**353**	**18s**
FrameFlow-guid.	**20**	192	**18s**
RFdiffusion	**20**	141	50s
TDS	19	217	117s

Tab. 1 provides the number of motifs each method solves – which means at least one designable scaffold is sampled – and the number of total designable clusters sampled across all motifs. Here we see each method can solve 19-20 solves motifs, but FrameFlow-amortization can achieve nearly double the number of unique scaffolds (clusters) as RFdiffusion. FrameFlow-amortization outperforms FrameFlow-guidance on diversity. A potential reason for the improved diversity is the use of SE(3) flow matching in the unconditional model whereas TDS uses SE(3) diffusion (Yim et al., 2023b). Bose et al. (2023) found SE(3) flow matching to provide far better designability and diversity than its diffusion counterpart. Empirically, it is known flow matching outperforms diffusion on Riemannian manifolds (Chen & Lipman, 2023).

In the last column we give the number of seconds to sample a length 100 protein on a A6000 Nvidia GPU with each method. Both FrameFlow methods are significantly faster than RFdiffusion and TDS. TDS is notably slower since its run time scales with its number of particles. We conclude that FrameFlow-amortization matches RFdiffusion and TDS on the number of solved motifs while achieving much higher diversity and faster inference.

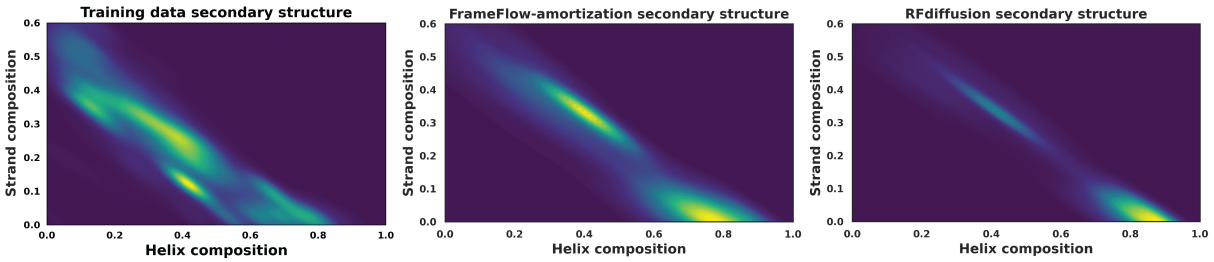


Figure 5: **Secondary structure analysis.** 2D kernel density plots of secondary structure composition of *designable* motif-scaffolds from FrameFlow-amortization and RFdiffusion. Here we see RFdiffusion tends to mostly generate helical scaffolds while FrameFlow-amortization gets much more scaffolds with strands.

Diversity analysis. To visualize the diversity of the scaffolds, Fig. 4 shows several of the clusters for motifs 1QJG, 1YCR, and 5TPN where FrameFlow can generate significantly more clusters than RFdiffusion. Each scaffold demonstrates a wide range of secondary structure elements across multiple lengths. To quantify this in more depth, Fig. 5 plot the helical and strand compositions (computed with DSSP (Kabsch & Sander, 1983)) of designable motif-scaffolds from FrameFlow-amortization compared to RFdiffusion. We see FrameFlow-amotization achieves a better spread of secondary structure components than RFdiffusion. A potential reason for RFdiffusion's overall lower diversity is due to its lack of secondary structure diversity – favoring to sample mostly helical structures. App. G provides additional analysis into the FrameFlow motif-scaffolding results. We conclude FrameFlow-amortization achieves much more structural diversity than RFdiffusion.

6 Discussion

In this work, we present two methods building on FrameFlow for tackling motif-scaffolding. These methods can be used with any flow-based model. First, with motif-amortization we adapt the training of FrameFlow to additionally be conditioned on the motif — in effect turning FrameFlow into a conditional generative model. Second, with motif guidance, we use an unconditionally trained FrameFlow for the task of motif-scaffolding though without any additional task-specific training. We empirically evaluated both approaches, FrameFlow-amortization and FrameFlow-guidance, on the motif-scaffolding benchmark from RFdiffusion where we find both methods achieve competitive results with state-of-the-art methods. Moreover, they are able to sample more unique scaffolds and achieve higher diversity. It is important to note amortization and guidance are complementary techniques. Amortization outperforms guidance but requires conditional training while guidance can use unconditional flow models without further training. Guidance generally performs worse due to approximation error in Eq. (12) from using an unconditional model in conditional task. We stress the need to report both success rate and diversity to detect when a model suffers from mode collapse. Lastly, we caveat that all our results and metrics are computational, which may not necessarily transfer to wet-lab success.

Future directions. We have extended FrameFlow for motif-scaffolding; further extensions include binder, enzyme, and symmetric design — all which RFdiffusion can currently achieve. For these capabilities, we require extending FrameFlow to handle multimeric proteins. While motif guidance does not outperform motif amortization, it is possible extending TDS to flow matching could close that gap. Related to guidance, one could explore conditioning mechanisms to control properties of the scaffold such as its secondary structure. We make use of a heuristic for Riemannian reconstruction guidance that may be further improved. Despite our progress, there still remains areas of improvement to achieve success in all 25 motifs in the benchmark.

References

Sarah Alamdari, Nitya Thakkar, Rianne van den Berg, Alex Xijie Lu, Nicolo Fusi, Ava Pardis Amini, and Kevin K Yang. Protein generation with evolutionary diffusion: sequence is all you need. *bioRxiv*, pp. 2023–09, 2023.

Michael S Albergo, Mark Goldstein, Nicholas M Boffi, Rajesh Ranganath, and Eric Vanden-Eijnden. Stochastic interpolants with data-dependent couplings. *arXiv preprint arXiv:2310.03725*, 2023.

Minkyung Baek, Ivan Anishchenko, Ian Humphreys, Qian Cong, David Baker, and Frank DiMaio. Efficient and accurate prediction of protein structure using rosettafold2. *bioRxiv*, pp. 2023–05, 2023.

Nathaniel R Bennett, Brian Coventry, Inna Goreshnik, Buwei Huang, Aza Allen, Dionne Vafeados, Ying Po Peng, Justas Dauparas, Minkyung Baek, Lance Stewart, et al. Improving de novo protein binder design with deep learning. *Nature Communications*, 14(1):2625, 2023.

Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, Talapady N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000.

Avishek Joey Bose, Tara Akhound-Sadegh, Kilian Fatras, Guillaume Huguet, Jarrid Rector-Brooks, Cheng-Hao Liu, Andrei Cristian Nica, Maksym Korablyov, Michael Bronstein, and Alexander Tong. Se (3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.

Ricky T. Q. Chen and Yaron Lipman. Riemannian Flow Matching on General Geometries, February 2023. URL http://arxiv.org/abs/2302.03660.

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

Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.

Bruno E Correia, John T Bates, Rebecca J Loomis, Gretchen Baneyx, Chris Carrico, Joseph G Jardine, Peter Rupert, Colin Correnti, Oleksandr Kalyuzhniy, Vinayak Vittal, Mary J Connell, Eric Stevens, Alexandria Schroeter, Man Chen, Skye Macpherson, Andreia M Serra, Yumiko Adachi, Margaret A Holmes, Yuxing Li, Rachel E Klevit, Barney S Graham, Richard T Wyatt, David Baker, Roland K Strong, James E Crowe, Jr, Philip R Johnson, and William R Schief. Proof of principle for epitope-focused vaccine design. *Nature*, 507(7491):201–206, 2014.

Quan Dao, Hao Phung, Binh Nguyen, and Anh Tran. Flow matching in latent space. *arXiv preprint arXiv:2307.08698*, 2023.

J. Dauparas, I. Anishchenko, N. Bennett, H. Bai, R. J. Ragotte, L. F. Milles, B. I. M. Wicky, A. Courbet, R. J. de Haas, N. Bethel, P. J. Y. Leung, T. F. Huddy, S. Pellock, D. Tischer, F. Chan, B. Koepnick, H. Nguyen, A. Kang, B. Sankaran, A. K. Bera, N. P. King, and D. Baker. Robust deep learning-based protein sequence design using ProteinMPNN. *Science*, 378(6615):49–56, 2022.

Kieran Didi, Francisco Vargas, Simon V Mathis, Vincent Dutordoir, Emile Mathieu, Urszula J Komorowska, and Pietro Lio. A framework for conditional diffusion modelling with applications in motif scaffolding for protein design. *arXiv preprint arXiv:2312.09236*, 2023.

James Dunbar, Konrad Krawczyk, Jinwoo Leem, Terry Baker, Angelika Fuchs, Guy Georges, Jiye Shi, and Charlotte M Deane. Sabdab: the structural antibody database. *Nucleic acids research*, 42(D1):D1140–D1146, 2014.

Alex Herbert and MJE Sternberg. MaxCluster: a tool for protein structure comparison and clustering. 2008.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models, 2022.

John B Ingraham, Max Baranov, Zak Costello, Karl W Barber, Wujie Wang, Ahmed Ismail, Vincent Frappier, Dana M Lord, Christopher Ng-Thow-Hing, Erik R Van Vlack, et al. Illuminating protein space with a programmable generative model. *Nature*, pp. 1–9, 2023.

Lin Jiang, Eric A Althoff, Fernando R Clemente, Lindsey Doyle, Daniela Rothlisberger, Alexandre Zanghellini, Jasmine L Gallaher, Jamie L Betker, Fujie Tanaka, Carlos F Barbas III, Donald Hilvert, Kendal N Houk, Barry L. Stoddard, and David Baker. De novo computational design of retro-aldol enzymes. *Science*, 319(5868):1387–1391, 2008.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules*, 22(12): 2577–2637, 1983.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Leon Klein, Andreas Krämer, and Frank Noé. Equivariant flow matching. *arXiv preprint arXiv:2306.15030*, 2023.

Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International conference on machine learning*, pp. 5361–5370. PMLR, 2020.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *International Conference on Learning Representations*, 2023.

Dmitry I Nikolayev and Tatjana I Savyolov. Normal distribution on the rotation group SO(3). *Textures and Microstructures*, 29, 1970.

Ashwini Pokle, Matthew J Muckley, Ricky TQ Chen, and Brian Karrer. Training-free linear image inversion via flows. *arXiv preprint arXiv:2310.04432*, 2023.

Erik Procko, Geoffrey Y Berguig, Betty W Shen, Yifan Song, Shani Frayo, Anthony J Convertine, Daciana Margineantu, Garrett Booth, Bruno E Correia, Yuanhua Cheng, William R Schief, David M Hockenbery, Oliver W Press, Barry L Stoddard, Patrick S Stayton, and David Baker. A computationally designed inhibitor of an Epstein-Barr viral BCL-2 protein induces apoptosis in infected cells. *Cell*, 157(7):1644–1656, 2014.

Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 conference proceedings*, pp. 1–10, 2022.

Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*. Cambridge University Press, 1 edition, April 2019. ISBN 978-1-108-18673-5 978-1-316-51008-7 978-1-316-64946-6. doi: 10.1017/9781108186735.

Neta Shaul, Ricky TQ Chen, Maximilian Nickel, Matthew Le, and Yaron Lipman. On kinetic optimal probability paths for generative models. In *International Conference on Machine Learning*, pp. 30883–30907. PMLR, 2023.

Justin B Siegel, Alexandre Zanghellini, Helena M Lovick, Gert Kiss, Abigail R Lambert, Jennifer L StClair, Jasmine L Gallaher, Donald Hilvert, Michael H Gelb, Barry L Stoddard, Kendall N Houk, Forrest E Michael, and David Baker. Computational design of an enzyme catalyst for a stereoselective bimolecular Diels-Alder reaction. *Science*, 329(5989):309–313, 2010.

Jiaming Song, Arash Vahdat, Morteza Mardani, and Jan Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*, 2022.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. Solving inverse problems in medical imaging with score-based generative models. *arXiv preprint arXiv:2111.08005*, 2021.

Brian L Trippe, Jason Yim, Doug Tischer, David Baker, Tamara Broderick, Regina Barzilay, and Tommi Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

Jue Wang, Sidney Lisanza, David Juergens, Doug Tischer, Ivan Anishchenko, Minkyung Baek, Joseph L Watson, Jung Ho Chun, Lukas F Milles, Justas Dauparas, Marc Exposit, Wei Yang, Amijai Saragovi, Sergey Ovchinnikov, and David A. Baker. Deep learning methods for designing proteins scaffolding functional sites. *bioRxiv*, 2021.

Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, pp. 1–3, 2023.

Luhuan Wu, Brian L Trippe, Christian A Naesseth, David M Blei, and John P Cunningham. Practical and asymptotically exact conditional sampling in diffusion models. *arXiv preprint arXiv:2306.17775*, 2023.

Kevin K Yang, Zachary Wu, and Frances H Arnold. Machine-learning-guided directed evolution for protein engineering. *Nature methods*, 16(8):687–694, 2019.

Jason Yim, Andrew Campbell, Andrew YK Foong, Michael Gastegger, José Jiménez-Luna, Sarah Lewis, Victor Garcia Satorras, Bastiaan S Veeling, Regina Barzilay, Tommi Jaakkola, et al. Fast protein backbone generation with se (3) flow matching. *arXiv preprint arXiv:2310.05297*, 2023a.

Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023b.

Qinqing Zheng, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky TQ Chen. Guided flows for generative modeling and decision making. *arXiv preprint arXiv:2311.13443*, 2023.

Appendix

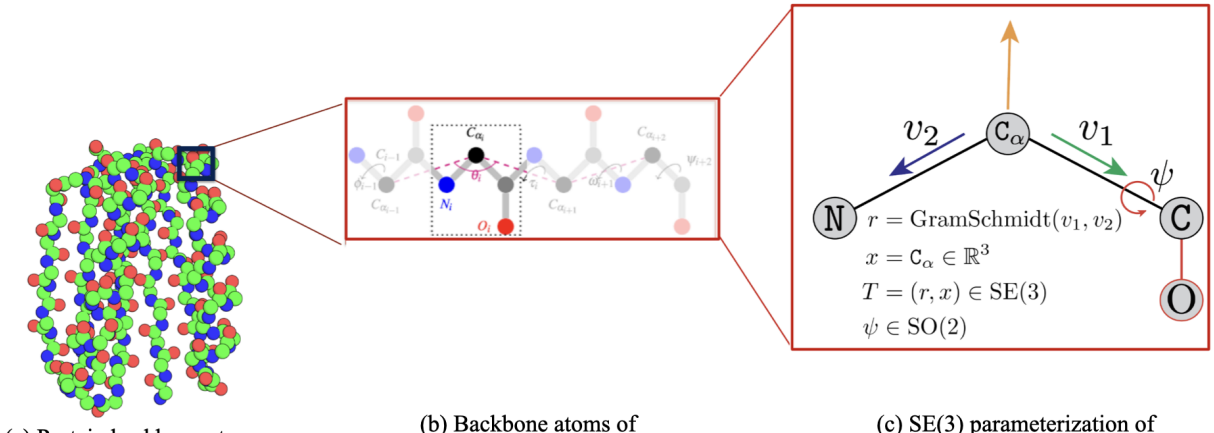
A Organisation of appendices

The appendix is organized as follows. App. B provides details and derivations for FrameFlow (Yim et al., 2023a) that we introduce in Sec. 2.2. App. D provides derivation of motif guidance used in Sec. 3.2. Designability is an important metric in our experients, so we provide a description of it in App. E. Lastly, we include additional results on unconditional generation App. F and motif-scaffolding App. G.

B FrameFlow details

B.1 Backbone SE(3) representation

A protein can be described by its sequence of residues, each of which takes on a discrete value from a vocabulary of amino acids, as well as the 3D structure based on the positions of atoms within each residue. The 3D structure in each residue can be separated into the backbone and side-chain atoms with the composition of backbone atoms being constant across all residues while the side-chain atoms vary depending on the amino acid assignment. For this reason, FrameFlow and previous SE(3) diffusion models (Watson et al., 2023; Yim et al., 2023b) only model the backbone atoms with the amino acids assumed to be unknown. A second model is typically used to design the amino acids after the backbone is generated. Each residue's backbone atoms follows a repeated arrangement with limited degrees of freedom due to the rigidity of the covalent bonds. AlphaFold2 (AF2) (Jumper et al., 2021) proposed a SE(3) parameterization of the backbone atoms that we show in Fig. 6. AF2 uses a mapping of four backbone atoms to a single translation and rotation that reduces the degrees of freedom in the modeling. It is this SE(3) representation we use when modeling protein backbones. We refer to Appendix I of Yim et al. (2023b) for algorithmic details of mapping between elements of SE(3) and backbone atoms.



(a) Protein backbone atoms

(b) Backbone atoms of a single residue

(c) SE(3) parameterization of backbone atoms of a single residue

Figure 6: Backbone parameterization with SE(3). (a) Shows the full protein backbone atomic structure without side-chains. (b) Zooms in the backbone atoms of a single residue. Note the repeated arrangements of backbone atoms in each residue. (c) The transformation of turning each set of four backbone atoms into an element of SE(3).

B.2 SE(3) flow matching implementation

This section provides implementation details for SE(3) flow matching and FrameFlow. As stated in Sec. 2.1, $\text{SE}(3)^N$ can be characterized as the product manifold $\text{SE}(3)^N = \mathbb{R}^N \times \text{SO}(3)^N$. It follows that flow matching on $\text{SE}(3)^N$ is equivalent to flow matching on $\mathbb{R}^{3N}$ and $\text{SO}(3)^N$. We will parameterize backbones with N

residues $\mathbf{T} = (\mathbf{x}, \mathbf{r}) \in \mathrm{SE}(3)^N$ by translations $\mathbf{x} \in \mathbb{R}^{3N}$ and rotations $\mathbf{r} \in \mathrm{SO}(3)^N$. As a reminder, we use bold face for vectors of all the residue: $\mathbf{T} = [T^{(1)}, \dots, T^{(N)}]$, $\mathbf{x} = [x^{(1)}, \dots, x^{(N)}]$, $\mathbf{r} = [r^{(1)}, \dots, r^{(N)}]$.

Riemannian flow matching (Sec. 2.1) proceeds by defining the conditional flows,

$$x_t^{(n)} = (1-t)x_0^{(n)} + t x_1^{(n)}, \qquad r_t^{(n)} = \exp_{r_0^{(n)}}\left(t \log_{r_0^{(n)}}(r_1^{(n)})\right), \tag{16}$$

for each residue $n \in \{1, \dots, N\}$. As priors we use $x_0^{(n)} \sim \overline{\mathcal{N}}(0, I_3)$ and $r_0^{(n)} \sim \mathcal{U}(\mathrm{SO}(3))$. $\overline{\mathcal{N}}(0, I_3)$ is the isotropic Gaussian in 3D with centering where each sample is centered to have zero mean – this is important for equivariance later on. $\mathcal{U}(\mathrm{SO}(3))$ is the uniform distribution over SO(3). The end points $x_1^{(n)}$ and $r_1^{(n)}$ are samples from the data distribution p_1.

Eq. (16) uses the geodesic path with linear interpolation; however, alternative conditional flows can be used (Chen & Lipman, 2023). A special property of SO(3) is that $\exp_{r_0}$ and $\log_{r_0}$ can be computed in closed form using the well known Rodrigues' formula. The corresponding conditional vector fields are

$$u_{\mathbb{R}}^{(n)}(x_t^{(n)}, t | x_1^{(n)}) = \frac{x_1^{(n)} - x_t^{(n)}}{1-t}, \qquad u_{\mathrm{SO}(3)}^{(n)}(r_t^{(n)}, t | r_1^{(n)}) = \frac{\log_{r_t^{(n)}}(r_1^{(n)})}{1-t}. \tag{17}$$

We train neural networks to regress the conditional vector fields through the following parameterization,

$$\hat{v}_{\mathbb{R}}^{(n)}(\mathbf{T}_t, t) = \frac{\hat{x}_1^{(n)}(\mathbf{T}_t) - x_t^{(n)}}{1-t}, \quad \hat{v}_{\mathrm{SO}(3)}^{(n)}(\mathbf{T}_t, t) = \frac{\log_{r_t^{(n)}}(\hat{r}_1^{(n)}(\mathbf{T}_t))}{1-t}, \tag{18}$$

where the neural network outputs the *denoised* predictions $\hat{x}_1^{(n)}$ and $\hat{r}_1^{(n)}$ while using the noised backbone $\mathbf{T}_t$ as input. We now modify the loss from Eq. (5) with practical details from FrameFlow,

$$\mathcal{L}_{\mathrm{SE}(3)} = \mathbb{E}_{\mathcal{U}(t;0,1), p_1(\mathbf{T}_1), p_0(\mathbf{T}_0)}\left[\mathcal{L}_{\mathbb{R}}(\mathbf{T}_t, \mathbf{T}_1, t) + 2\mathcal{L}_{\mathrm{SO}(3)}(\mathbf{T}_t, \mathbf{T}_1, t) + \mathbb{1}(t > 0.5)\mathcal{L}_{\mathrm{aux}}(\mathbf{T}_t, \mathbf{T}_1, t)\right] \tag{19}$$

$$\mathcal{L}_{\mathbb{R}}(\mathbf{T}_t, \mathbf{T}_1, t) = \|\mathbf{u}_{\mathbb{R}}(\mathbf{x}_t | \mathbf{x}_1, t) - \hat{\mathbf{v}}_{\mathbb{R}}(\mathbf{T}_t, t)\|_{\mathbb{R}}^2 = \frac{\|\mathbf{x}_1 - \hat{\mathbf{x}}_1\|_{\mathbb{R}}^2}{(1 - \min(t, 0.9))^2} \tag{20}$$

$$\mathcal{L}_{\mathrm{SO}(3)}(\mathbf{T}_t, \mathbf{T}_1, t) = \left\|\mathbf{u}_{\mathrm{SO}(3)}(\mathbf{r}_t | \mathbf{r}_1, t) - \hat{\mathbf{v}}_{\mathrm{SO}(3)}(\mathbf{T}_t, t)\right\|_{\mathrm{SO}(3)}^2 = \frac{\left\|\log_{\mathbf{r}_t^{(n)}}(\mathbf{r}_1^{(n)}) - \log_{\mathbf{r}_t^{(n)}}(\hat{\mathbf{r}}_1^{(n)}(\mathbf{T}_t))\right\|_{\mathrm{SO}(3)}^2}{(1 - \min(t, 0.9))^2}. \tag{21}$$

We up weight the SO(3) loss $\mathcal{L}_{\mathrm{SO}(3)}$ such that it is on a similar scale as the translation loss $\mathcal{L}_{\mathbb{R}}$. Eq. (20) is simplified to be a loss directly on the denoised predictions. Both Eq. (20) and Eq. (21) have modified denominators $(1 - \min(t, 0.9))^{-2}$ instead of $(1-t)^{-1}$ to avoid the loss blowing up near $t \approx 1$. In practice, we sample t uniformly from $\mathcal{U}[\epsilon, 1]$ for small ϵ. Lastly, $\mathcal{L}_{\mathrm{aux}}$ is taken from section 4.2 in Yim et al. (2023b) where they apply a RMSD loss over the full backbone atom positions and pairwise distances. We found using $\mathcal{L}_{\mathrm{aux}}$ for all $t > 0.5$ to be helpful. The remainder of this section goes over additional details in FrameFlow.

Alternative SO(3) prior. Yim et al. (2023a) reported using the IGSO3($\sigma = 1.5$) prior (Nikolayev & Savyolov, 1970) for SO(3) instead of $\mathcal{U}(\mathrm{SO}(3))$ lead to improved performance. The choice of $\sigma = 1.5$ will shift the r_0 samples away from π where near degenerate solutions can arise in the geodesic. We follow using IGSO3($\sigma = 1.5$) for training while using the $\mathcal{U}(\mathrm{SO}(3))$ prior for sampling.

Pre-alignment. Following (Klein et al., 2023) and Shaul et al. (2023), we pre-align samples from the prior and the data by using the Kabsch algorithm to align the noise with the data to remove any global rotation that results in a increased kinetic energy of the ODE. Specifically, for translation noise $\mathbf{x}_0 \sim \mathcal{N}(0, \mathrm{I}_3)^N$ and data $\mathbf{x}_1 \sim p_1$ where $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^{3 \times N}$ we solve $r^* = \arg\min_{r \in \mathrm{SO}(3)} \|r\mathbf{x}_0 - \mathbf{x}_1\|_{\mathbb{R}}^2$ and use the *aligned* noise $r^*\mathbf{x}_0$ during training. Yim et al. (2023a) found this to aid in training efficiency which we adopt.

Symmetries. We perform all modelling within the zero center of mass (CoM) subspace of $\mathbb{R}^{N\times 3}$ as in Yim et al. (2023b). This entails simply subtracting the CoM from the prior sample $\mathbf{x}_0$ and all datapoints $\mathbf{x}_1$. As $\mathbf{x}_t$ is a linear interpolation between the noise sample and data, $\mathbf{x}_t$ will have 0 CoM also. This guarantees that the distribution of sampled frames that the model generates is SE(3)-invariant. To see this, note that the prior distribution is SE(3)-invariant and the learned vector field $\mathbf{v}_{\mathrm{SE}(3)}$ is equivariant because we use an SE(3)-equivariant architecture. Hence by Köhler et al. (2020), the push-forward of the prior under the flow is invariant.

Auxiliary losses. We use the same auxiliary losses in (Yim et al., 2023a).

SO(3) inference scheduler. The conditional flow in Eq. (16) uses a constant linear interpolation along the geodesic path where the distance of the current point x to the endpoint x_1 is given by a pre-metric $d_g : \mathcal{M}\times\mathcal{M}\to\mathbb{R}$ induced by the Riemannian metric g on the manifold. To see this, we first recall the general form of the conditional vector field with $x, x_1 \in \mathcal{M}$ is given as follows (Chen & Lipman, 2023),

$$u_t(x|x_1) = \frac{\mathrm{d}\log\kappa(t)}{\mathrm{d}t} d(x,x_1)\frac{\nabla d(x,x_1)}{\|\nabla d(x,x_1)\|^2} \tag{22}$$

$$= \frac{\mathrm{d}\log\kappa(t)}{\mathrm{d}t}\frac{\nabla d(x,x_1)^2}{2\|\nabla d(x,x_1)\|^2} \tag{23}$$

$$= \frac{\mathrm{d}\log\kappa(t)}{\mathrm{d}t}\frac{-\log_x(x_1)}{\|\nabla d(x,x_1)\|^2} \tag{24}$$

$$= \frac{-\mathrm{d}\log\kappa(t)}{\mathrm{d}t}\log_x(x_1), \tag{25}$$

with $\kappa(t)$ a monotonically decreasing differentiable function satisfying $\kappa(0)=1$ and $\kappa(1)=0$, referred as the *interpolation rate* [7]. Then plugging in a the linear schedule $\kappa(t)=1-t$, we recover Eq. (17)

$$u_t(x|x_1) = \frac{-\mathrm{d}\log\kappa(t)}{\mathrm{d}t}\log_x(x_1) = \frac{1}{1-t}\log_x(x_1). \tag{26}$$

However, we found this interpolation rate to perform poorly for SO(3) for inference time. Instead, we utilize an exponential scheduler $\kappa(t)=e^{-ct}$ for some constant c. The intuition being that for high c, the rotations accelerate towards the data faster than the translations which evolve according to the linear schedule. The SO(3) conditional flow in Eq. (16) and vector field in Eq. (17) become the following with the exponential schedule,

$$r_t = \exp_{r_0}\left(\left(1-e^{-ct}\right)\log_{r_0}(r_1)\right) \tag{27}$$

$$v_r^{(n)} = c\log_{r_t^{(n)}}\left(\hat{r}_1^{(n)}\right). \tag{28}$$

We find $c=10$ or 5 to work well and use $c=10$ in our experiments. Interestingly, we found the best performance when $\kappa(t)=1-t$ was used for SO(3) during training while $\kappa(t)=e^{-ct}$ is used during inference. We found using $\kappa(t)=e^{-ct}$ during training made training too easy with little learning happening.

The vector field in Eq. (28) matches the vector field in FoldFlow when inference *annealing* is performed (Bose et al., 2023). However, their choice of scaling was attributed to normalizing the predicted vector field rather than the schedule. Indeed they proposed to linearly scale up the learnt vector field via $\lambda(t)=(1-t)c$ at sampling time, i.e. to simulate the following ODE:

$$\mathrm{d}r_t = \lambda(t)v(r_t,t)\mathrm{d}t.$$

However, as hinted at earlier, this is equivalent to using at sampling time a different vector field $\tilde{v}(r_t,t)$—induced by an *exponential* schedule $\tilde{\kappa}(t)=e^{-ct}$—instead of the *linear* schedule $\kappa(t)=1-t$ (that the neural

[7] $\kappa(t)$ acts as a scheduler that determines the rate at which $d(\cdot|x_1)$ decreases, since we have that ϕ_t decreases $d(\cdot,x_1)$ according to $d(\phi_t(x_0|x_1),x_1)=\kappa(t)d(x_0,x_1)$(Chen & Lipman, 2023).

network $\hat{r}_1^\theta$ was trained with). Indeed we have

$$\tilde{v}(r_t, t) = -\partial_t \log \tilde{\kappa}(t) \log_{r_t}(\hat{r}_1) = -\frac{-\partial_t \log \tilde{\kappa}(t)}{-\partial_t \log \kappa(t)} \partial_t \log \kappa(t) \log_{r_t}(\hat{r}_1) \tag{29}$$

$$= -c(1-t)\partial_t \log \kappa(t) \log_{r_t}(\hat{r}_1) = c(1-t)\ v(r_t, t) = \lambda(t)\ v(r_t, t). \tag{30}$$

C Data augmentation

Algorithm 1 Motif-scaffolding data augmentation

Require: Protein backbone $\mathbf{T}$; Min and max motif percent $\gamma_{\min} = 0.05$, $\gamma_{\max} = 0.5$.

$s \sim \text{Uniform}\{\lfloor N \cdot \gamma_{\min} \rfloor, \dots, \lfloor N \cdot \gamma_{\max} \rfloor\}$ ▷ Sample maximum motif size.
$m \sim \text{Uniform}\{1, \dots, s\}$ ▷ Sample maximum number of motifs.
$\mathbf{T}^M \leftarrow \emptyset$
for $i \in \{1, \dots, m\}$ **do**
  $j \sim \text{Uniform}\{1, \dots, N\} \setminus \mathbf{T}^M$ ▷ Sample location for each motif
  $\ell \sim \text{Uniform}\{1, \dots, s - m + i - |\mathbf{T}^M|\}$ ▷ Sample length of each motif.
  $\mathbf{T}^M \leftarrow \mathbf{T}^M \cup \{T_j, \dots, T_{\min(j+\ell, N)}\}$ ▷ Append to existing motif.
end for
$\mathbf{T}^S \leftarrow \{T_1, \dots, T_N\} \setminus \mathbf{T}^M$ ▷ Assign rest of residues as the scaffold
return $\mathbf{T}^M, \mathbf{T}^S$

D Motif guidance details

For the sake of completeness, we derive in this section the guidance term in Eq. (8) for the flow matching setting. In particular, we want to derive the conditional vector field $v(x_t, t|y)$ in terms of the unconditional vector field $v(x_t, t)$ and the correction term $\nabla \log p_t(y|x_t)$. Beware, in the following we adopt the time notation from diffusion models, i.e. $t = 0$ for denoised data to $t = 1$ for fully noised data. We therefore need to swap $t \to 1 - t$ in the end results to revert to the flow matching notations.

Let's consider the process associated with the following noising stochastic differential equation (SDE)

$$\mathrm{d}x_t = f(x_t, t)\mathrm{d}t + g(t)\mathrm{dB}_t \tag{31}$$

which admits the following time-reversal denoising process

$$\mathrm{d}x_t = \left[f(x_t, t) - g(t)^2 \nabla \log p_t(x_t)\right] \mathrm{d}t + g(t)\mathrm{dB}_t. \tag{32}$$

Thanks to the Fokker-Planck equation, we know that the the following ordinary differential equation admits the same marginal as the SDE Eq. (32):

$$\begin{aligned} \mathrm{d}x_t &= \left[f(x_t, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(x_t)\right] \mathrm{d}t \\ &= v(x_t, t)\mathrm{d}t. \end{aligned} \tag{33}$$

with $v(x_t, t)$ being the probability flow vector field.

Now, conditioning on some observation y, we have

$$\begin{aligned} \mathrm{d}x_t &= v(x_t, t|y)\mathrm{d}t \\ &= \left[f(x_t, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(x_t|y)\right] \mathrm{d}t \\ &= \left[f(x_t, t) - \frac{1}{2} g(t)^2 \left(\nabla \log p_t(x_t) + \nabla \log p_t(y|x_t)\right)\right] \mathrm{d}t \\ &= \left[v(x_t, t) - \frac{1}{2} g(t)^2 \nabla \log p_t(y|x_t)\right] \mathrm{d}t. \end{aligned} \tag{34}$$

Eq. (34) follows from the same reverse SDE theory of Eq. (32) except the initial state distribution is $p(x_t|y)$). The drift $f(x_t, t)$ and diffusion $g(t)$ coefficients are unchanged while only the score reflect the new initial distribution. More details can be found in App. I of Song et al. (2020). We only need to know $g(t)$ to adapt reconstruction guidance–which estimates $\nabla \log p_t(y|x_t)$–to the flow matching setting where we want to correct the vector field. Given a particular choice of interpolation x_t from flow matching, let's derive the associated $g(t)$.

Euclidean setting Assume x_0 is data and x_1 is noise, with $x_1 \sim \mathcal{N}(0, \mathrm{I})$. In Euclidean flow matching, we assume a linear interpolation $x_t = (1-t)x_0 + tx_1$. Conditioning on x_0, we have the following conditional marginal density $p_{t|0} = \mathcal{N}\left((1-t)x_0, t^2\mathrm{I}\right)$. Meanwhile, let's derive the marginal density $\tilde{p}_{t|0}$ induced by Eq. (31). Assuming a linear drift $f(x_t, t) = \mu(t)x_t$, we know that $\tilde{p}_{t|0}$ is Gaussian. Let's derive its mean $m_t = \mathbb{E}[x_t]$ and covariance $\Sigma_t = \mathrm{Cov}[x_t]$. We have that (Särkkä & Solin, 2019)

$$\frac{\mathrm{d}}{\mathrm{d}t} m_t = \mathbb{E}\left[f(x_t, t)\right] = \mu(t) m_t. \tag{35}$$

thus

$$\mathbb{E}[x_t] = \exp\left(\int_0^t \mu(s)\mathrm{d}s\right) x_0. \tag{36}$$

Additionally,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\Sigma_t &= \mathbb{E}\left[f(x_t, t)(m_t - x_t)^\top\right] + \mathbb{E}\left[f(x_t, t)^\top (m_t - x_t)\right] + g(t)^2\mathrm{I} && (37)\\ &= 2\mu(t)\Sigma_t + g(t)^2\mathrm{I}, && (38) \end{aligned}$$

Matching $\tilde{p}_{t|0}$ and $p_{t|0}$, we get

$$\begin{aligned} &\exp\left(\int_0^t \mu(s)\mathrm{d}s\right)\ x_0 = (1-t)x_0 && (39)\\ \Leftrightarrow\ &\int_0^t \mu(s)\mathrm{d}s = \ln(1-t) && (40)\\ \Leftrightarrow\ &\mu(t) = -\frac{1}{1-t} && (41) \end{aligned}$$

and

$$\begin{aligned} &2\mu(t)t^2 + g(t)^2 = 2t && (42)\\ \Leftrightarrow\ &-2\frac{1}{1-t}t^2 + g(t)^2 = 2t && (43)\\ \Leftrightarrow\ &g(t)^2 = 2t + 2\frac{1}{1-t}t^2 && (44)\\ \Leftrightarrow\ &g(t)^2 = \frac{2t}{1-t}. && (45) \end{aligned}$$

The equivalent SDE that gives the same marginals is Therefore, the following SDE gives the same conditional marginal as flow matching:

$$\mathrm{d}x_t = \frac{-1}{1-t}x_t\mathrm{d}t + \sqrt{\frac{2t}{1-t}}\mathrm{dB}_t. \tag{46}$$

SO(3) setting The conditional marginal density $\tilde{p}_{t|0}$ induced by Eq. (31) with zero drift $f(r_t, t) = 0$ is given by the IGSO(3) distribution (Yim et al., 2023b): $\tilde{p}_{t|0} = \mathrm{IGSO}(3)\left(r_t, r_0, t\right)$. We are not aware of a closed form formula for the variance of such a distribution.

On the flow matching side, we assume r_0 is data and r_1 is noise, with $r_1 \sim \mathcal{U}(\mathrm{SO}(3))$, and a geodesic interpolation $r_t = \exp_{r_0}(t\log_{r_0}(r_1))$. We posit that the induced conditional marginal $p_{t|0}$ is *not* an IGSO(3) distribution. As such, it appears non-trivial to derive the required equivalent diffusion coefficient $g(t)$ for SO(3). We therefore use as a heuristic the same $g(t)$ as for $\mathbb{R}^d$.

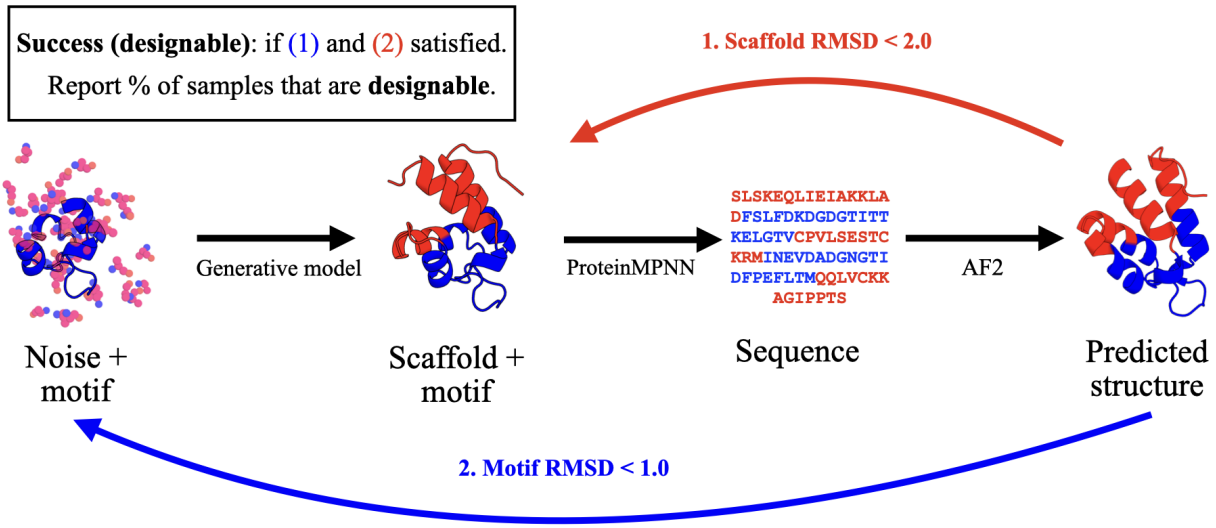


Figure 7: Schematic of computing motif-scaffolding designability. "Generative model" is a stand-in for the method used to generate scaffolds conditioned on the motif. From there, we use ProteinMPNN (Dauparas et al., 2022) to design the sequence then use AlphaFold2 (AF2) (Jumper et al., 2021) in predicting the structure of the sequence. The RMSD is calculated on the scaffold (blue) and motif (red) separately with alignments. A generated scaffold passes designability if the scaffold RMSD < 2.0 and motif RMSD < 1.0.

E Designability

We provide details of the designability metric for motif-scaffolding and unconditional backbone generation previously used in prior works (Watson et al., 2023; Wu et al., 2023). The quality of a backbone structure is nuanced and difficult to find a single metric for. One approach that has been proven reliable in protein design is using a highly accurate protein structure prediction network to recapitulate the structure after the sequence is inferred. Prior works (Bennett et al., 2023; Wang et al., 2021) found the best method for filtering backbones to use in wet-lab experiments was the combination of ProteinMPNN (Dauparas et al., 2022) to generate the sequences and AlphaFold2 (AF2) (Jumper et al., 2021) to recapitulate the structure. We choose to use the same procedure in determining the computational success of our backbone samples which we describe next. As always, we caveat these results are computational and may not transfer to wet-lab validation. While ESMFold (Jumper et al., 2021) may be used in place of AF2, we choose to follow the setting of RFdiffusion as close as possible.

We refer to *sampled backbones* as backbones generated from our generative model. Following RFdiffusion, we use ProteinMPNN at temperature 0.1 to generate 8 sequences for each backbone in motif-scaffolding and unconditional backbone generation. In motif-scaffolding, the motif amino acids are kept fixed – ProteinMPNN only generates amino acids for the scaffold. The *predicted backbone* of each sequence is obtained with the fourth model in the five model ensemble used in AF2 with 0 recycling, no relaxation, and no multiple sequence alignment (MSA) – as in the MSA is only populated with the query sequence. Fig. 7 provides a schematic of how we compute designability for motif-scaffolding. A sampled backbone is successful or referred to as designable based on the following criterion depending on the task:

- **Unconditional backbone generation**: successful if the Root Mean Squared Deviation (RMSD) of all the backbone atoms is < 2.0Åafter global alignment of the Carbon alpha positions.
- **Motif-scaffolding**: successful if the RMSD of motif atoms is < 1Åafter alignment on the motif Carbon alpha positions. Additionally, the RMSD of the scaffold atoms must be < 2Åafter alignment on the scaffold Carbon alpha positions.

F FrameFlow unconditional results

We present backbone generation results of the unconditional FrameFlow model used in FrameFlow-guidance. We do not perform an in-depth analysis since this task is not the focus of our work. Characterizing the backbone generation performance ensures we are using a reliable unconditional model for motif-scaffolding. We evaluate the unconditionally trained FrameFlow model by sampling 100 samples from lengths 70, 100, 200, and 300 as done in RFdiffusion. The results are shown in Tab. 2. We find that FrameFlow achieves slightly worse designability while achieving improved novelty. We conclude that FrameFlow is able to achieve strong unconditional backbone generation results that are on par with a current state-of-the-art unconditional diffusion model RFdiffusion. We perform secondary structure analysis of the unconditional samples in Fig. 8.

Table 2: Unconditional generation metrics.

Method	Des.(↑)	Div. (↑)	Nov. (↓)
FrameFlow	0.86	155	**0.61**
RFdiffusion	**0.89**	**159**	0.65

We find FrameFlow has a tendency to sample more alpha helical structures than the data distribution but still has roughly the same coverage of structures. Future work could investigate the cause of such helical tendency and improve the secondary structure sample distribution.

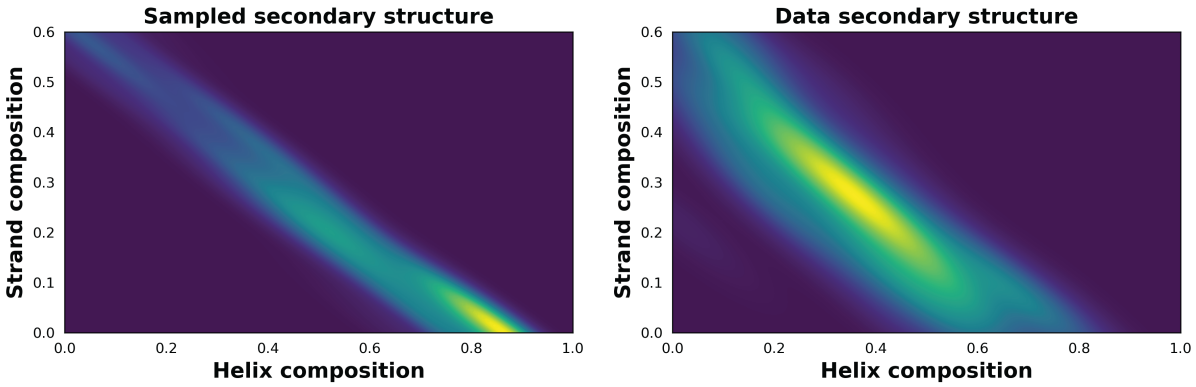


Figure 8: Secondary structure analysis of unconditional samples. Using FrameFlow we sample 100 proteins for each length 70, 100, 200, and 300. The 2D kernel density estimation plot of the secondary structure composition is shown on left. On the right, we show the secondary structure composition of all length 70, 100, 200, and 300 proteins in the training set. We find FrameFlow has a tendency to sample more alpha helical structures than the data distribution.

G FrameFlow motif-scaffolding analysis

In this section, we provide additional analysis into the motif-scaffolding results in Sec. 5.2. Our focus is on analyzing the motif-scaffolding with FrameFlow: motif amortization and guidance. The first analysis is the empirical cumulative distribution functions (ECDF) of the motif and scaffold RMSD shown in Fig. 9. We find that the main advantage of amortization is in having a higher percent of samples passing the motif RMSD threshold compared to the scaffold RMSD. Amortization has better scaffold RMSD but the gap is smaller than motif RMSD. The ECDF curves are roughly the same for both methods.

Tab. 3 shows the average pairwise TM-score for all designable scaffolds per motif. We report this for each method where we find our FrameFlow approaches get the lowest average pairwise TM-score in 19 out of 24 motifs. The average pairwise TM-score is meant to complement the cluster criterion for diversity in case where clustering leads to pathological behaviors. Both metrics have their strengths and weaknesses but together help provide more details on sample diversity.

Lastly, we visualize samples from FrameFlow-amortization on each motif in the benchmark. As noted in Sec. 5.2, amortization is able to solve 21 out of 24 motifs in the benchmark. In Fig. 10, we visualize the generated scaffolds that are closest to passing designability for the 3 motifs it is unable to solve. We find the failure to be in the motif RMSD being over 1ÅRMSD. However, for 4JHW, 7MRX_60, and 7MRX_128 the motif RMSDs are 1.2, 1.1, and 1.7 respectively. This shows amortization is very close to solve all motifs in the benchmark. In Fig. 11, we show designable scaffolds for each of the 21 motifs that amortization solves. We highlight the diverse range of motifs that can be solved as well as diverse scaffolds.

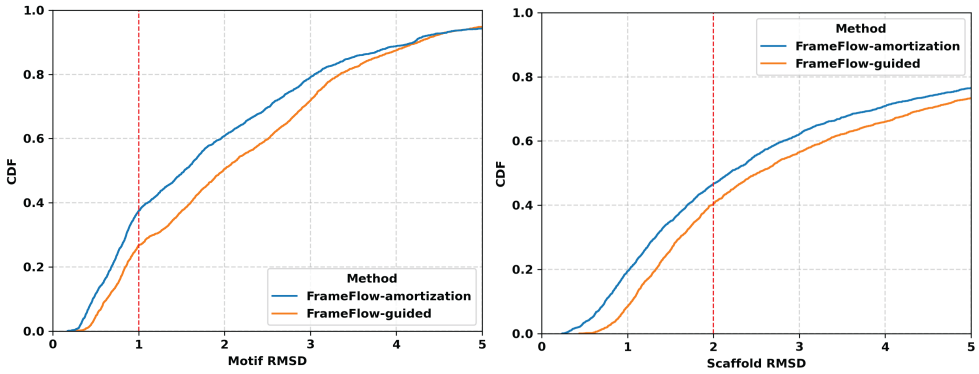


Figure 9: Empirical cumulative density plot of designability RMSD over the motif (left) and scaffold (right) for FrameFlow-amortization (blue line) and FrameFlow-guidance (orange line).

Table 3: Under each method name are average TM-score over all designable scaffolds for each motif. Lower is better which indicates more dissimilar pairwise scaffolds. N/A indicates no designable scaffolds were sampled. We find our FrameFlow approaches have the lowest average TM-scores among all methods.

Motif	FrameFlow-amortization	FrameFlow-guidance	TDS	RFdiffusion
6E6R_med	**0.3**	0.31	0.36	0.39
2KL8	0.95	**0.86**	0.88	0.98
4ZYP	0.55	**0.44**	N/A	N/A
5WN9	0.53	**0.45**	0.48	N/A
5TRV_short	0.48	**0.42**	0.46	0.66
7MRX_60	N/A	N/A	**0.29**	0.59
6EXZ_short	0.48	0.49	0.47	**0.35**
1YCR	0.34	**0.3**	0.4	0.48
5IUS	**0.6**	N/A	N/A	0.73
6E6R_short	0.37	**0.35**	0.39	0.41
3IXT	**0.4**	0.47	0.46	0.62
7MRX_85	N/A	**0.36**	N/A	0.56
1QJG	0.34	0.44	**0.33**	N/A
1BCF	0.76	0.69	**0.5**	0.83
5TRV_med	**0.34**	0.37	0.38	0.43
5YUI	N/A	N/A	N/A	N/A
5TPN	0.48	**0.54**	N/A	0.61
1PRW	0.75	**0.66**	N/A	0.76
6EXZ_med	0.37	0.38	**0.36**	0.49
5TRV_long	**0.3**	0.31	N/A	0.39
4JHW	N/A	N/A	N/A	N/A
7MRX_128	N/A	N/A	N/A	**0.5**
6E6R_long	**0.28**	**0.28**	0.46	0.35
6EXZ_long	**0.3**	**0.3**	0.38	0.38

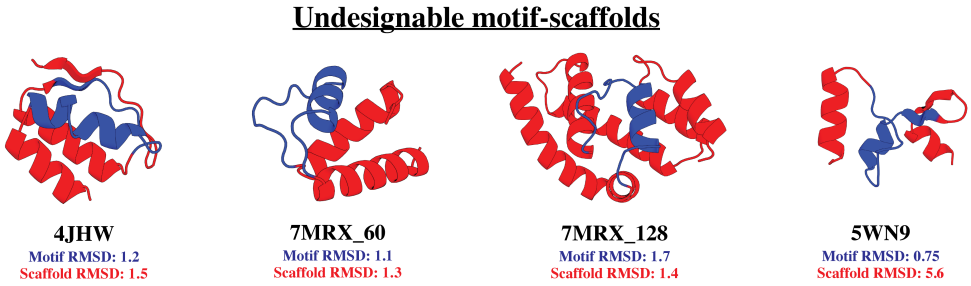


Figure 10: Closest motif-scaffolds from FrameFlow-amortization on the three motifs it fails to solve. We find the > 1.0Åmotif RMSD is the reason for the method failing to pass designability.

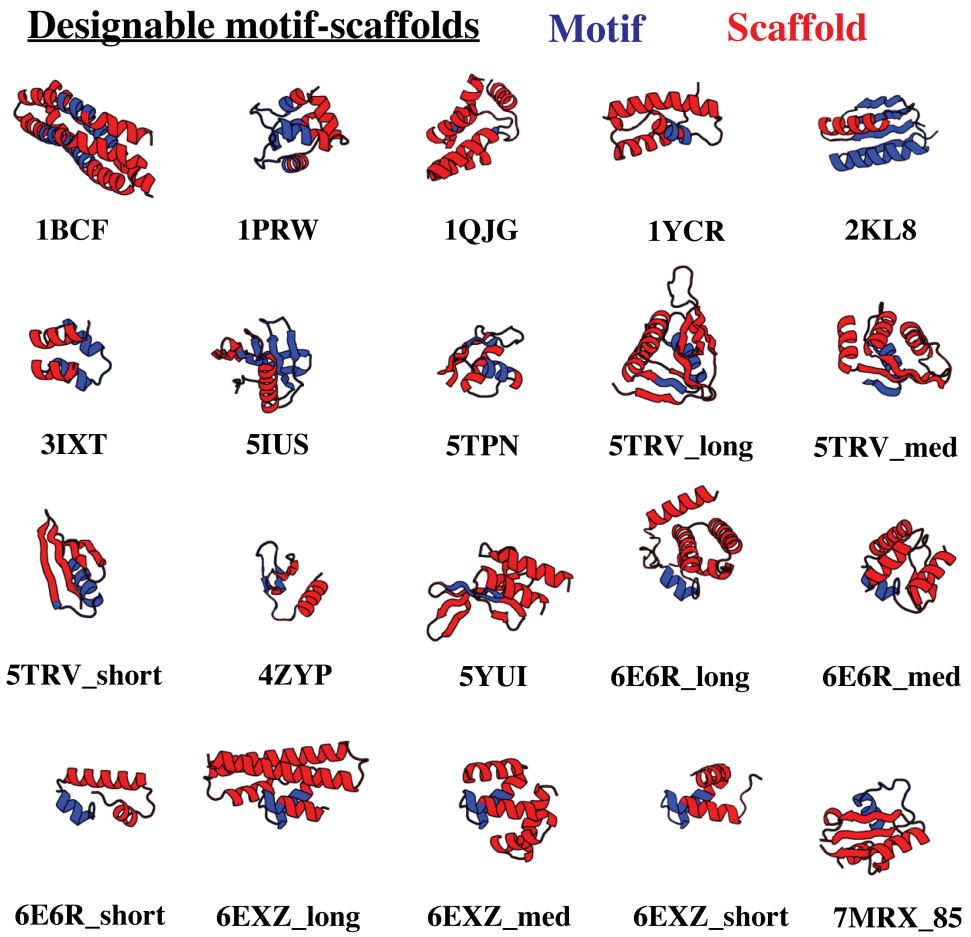


Figure 11: Designable motif-scaffolds from FrameFlow-amortization on 20 out of 24 motifs in the motif-scaffolding benchmark.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Improved motif-scaffolding with SE(3) flow matching
Publication Status	Published
Publication Details	Yim, J., Campbell, A., Mathieu, E., Foong, A.Y., Gastegger, M., Jiménez-Luna, J., Lewis, S., Satorras, V.G., Veeling, B.S., Noé, F. and Barzilay, R., 2024. Improved motif-scaffolding with SE (3) flow matching. Transactions on Machine Learning Research

Student Confirmation

Student Name:	Andrew Campbell		
Contribution to the Paper	- Contributed to the code-base and ran experiments investigating the model parameterization - Wrote initial version of the diffusion-flow connection for use in guidance - Wrote draft sections of the initial manuscript including the SE(3) flow formalism		
Signature		Date	08-Jan-2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Arnaud Doucet, Senior Staff Research Scientist Google DeepMind			
Supervisor comments I agree with the description of the contributions.			
Signature		Date	09-Jan-2025

This completed form should be included in the thesis, at the end of the relevant chapter.

6

Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design

Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design

Andrew Campbell [*1] **Jason Yim** [*2] **Regina Barzilay** [2] **Tom Rainforth** [1] **Tommi Jaakkola** [2]

Abstract

Combining discrete and continuous data is an important capability for generative models. We present Discrete Flow Models (DFMs), a new flow-based model of discrete data that provides the missing link in enabling flow-based generative models to be applied to multimodal continuous and discrete data problems. Our key insight is that the discrete equivalent of continuous space flow matching can be realized using Continuous Time Markov Chains. DFMs benefit from a simple derivation that includes discrete diffusion models as a specific instance while allowing improved performance over existing diffusion-based approaches. We utilize our DFMs method to build a multimodal flow-based modeling framework. We apply this capability to the task of protein co-design, wherein we learn a model for jointly generating protein structure and sequence. Our approach achieves state-of-the-art co-design performance while allowing the same multimodal model to be used for flexible generation of the sequence or structure.

1. Introduction

Expanding the capabilities of generative models to handle discrete and continuous data, which we refer to as *multimodal*, is a fundamental problem to enable their widespread adoption in scientific applications (Wang et al., 2023). One such application requiring a multimodal generative model is protein co-design where the aim is to jointly generate continuous protein structures alongside corresponding discrete amino acid sequences (Shi et al., 2022). Proteins have been well-studied: the function of the protein is endowed through its structure while the sequence is the blueprint of how the structure is made. This interplay motivates jointly generating the structure and sequence rather than in isolation. To this end, the focus of our work is to develop a multimodal generative framework capable of co-design.

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020) have achieved state-of-the-art performance across multiple applications. They have potential as a multimodal framework because they can be defined on both continuous and discrete spaces (Hoogeboom et al., 2021; Austin et al., 2021). However, their sample time inflexibility makes them unsuitable for multimodal problems. On even just a single modality, finding optimal sampling parameters requires extensive re-training and evaluations (Karras et al., 2022). This problem is exacerbated for multiple modalities. On the other hand, flow-based models (Liu et al., 2023; Albergo & Vanden-Eijnden, 2023; Lipman et al., 2023) improve over diffusion models with a simpler framework that allows for superior performance through sampling flexibility (Ma et al., 2024). Unfortunately, our current inability to define a flow-based model on discrete spaces holds us back from a multimodal flow model.

We address this by introducing a novel flow-based model for discrete data named **Discrete Flow Models (DFMs)** and thereby unlock a complete framework for flow-based multimodal generative modeling. Our key insight comes from seeing that a discrete flow-based model can be realized using Continuous Time Markov Chains (CTMCs). DFMs are a new discrete generative modeling paradigm: less restrictive than diffusion, allows for sampling flexibility without re-training and enables simple combination with continuous state space flows to form multimodal flow models.

Fig. 1A provides an overview of DFMs. We first define a probability flow p_t that linearly interpolates from noise to data. We then generate new data by simulating a sequence trajectory x_t that follows p_t across time which requires training a denoising neural network with cross-entropy. The sequence trajectory could have many transitions or few, a property we term CTMC Stochasticity (Fig. 1B). Prior discrete diffusion models are equivalent to picking a specific stochasticity at training time, whereas we can adjust it at

[*]Equal contribution [1]Department of Statistics, University of Oxford, UK [2]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Massachusetts, USA. Correspondence to: Andrew Campbell <campbell@stats.ox.ac.uk>, Jason Yim <jyim@csail.mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

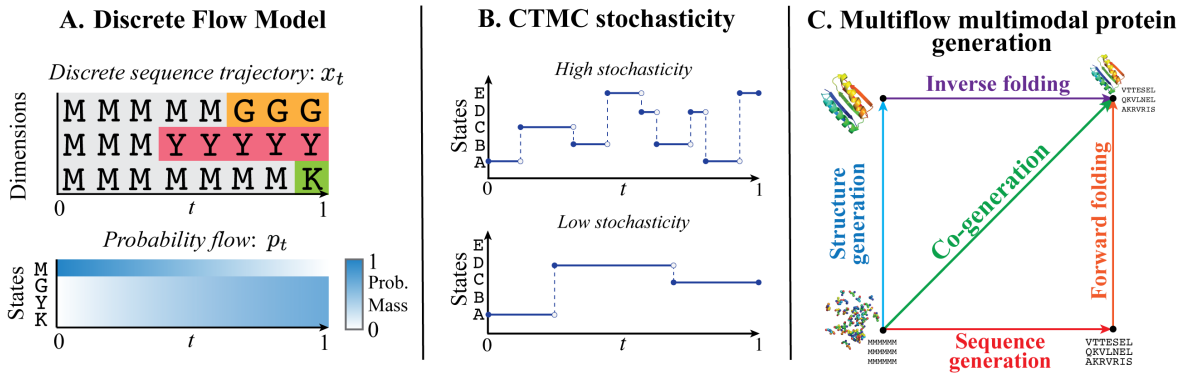


Figure 1. **Overview. (A.)** A DFM trajectory with masking over a 3-dim. sequence with 4 possible states. **(B.)** CTMC stochasticity controls the number of transitions in a sequence trajectory *while respecting the flow* p_t. Shown is a 1-dim. sequence with 5 states. **(C.)** Sampling with Multiflow can start from noise (bottom left) or with either the structure or sequence given (top left and bottom right). Any sampling tasks (structure/sequence generation, forward/inverse folding, co-generation) can be achieved with a single Multiflow model.

inference: enhancing sample quality and exerting control over sample distributional properties.

Using DFMs, we are then able to create a multimodal flow model by defining factorized flows for each data modality. We apply this capability to the task of protein co-design by developing a novel continuous structure and discrete sequence generative model named **Multiflow**. We combine a DFM for sequence generation and a flow-based structure generation method developed in Yim et al. (2023a). Previous multimodal approaches either generated only the sequence or only the structure and then used a prediction model to infer the remaining modality (see Sec. 5). Our *single* model can jointly generate sequence and structure while being able to condition on either modality.

In our experiments (Sec. 6), we first verify on small scale text data that DFMs outperform the discrete diffusion alternative, D3PM (Austin et al., 2021) through their expanded sample time flexibility. We then move to our main focus, assessing Multiflow's performance on the co-design task of jointly generating protein structure and sequence. Multiflow achieves state-of-the-art co-design performance while data distillation allows for obtaining state-of-the-art structure generation. We find CTMC stochasticity enables controlling sample properties such as secondary structure composition and diversity. Preliminary results on inverse and forward folding show Multiflow is a promising path towards a general-purpose protein generative model.

Our contributions are summarized as follows:

- We present Discrete Flow Models (DFMs), a novel discrete generative modeling method built through a CTMC simulating a probability flow.
- We combine DFMs with continuous flow-based methods to create a multimodal generative modeling framework.
- We use our multimodal framework to develop Multiflow, a state-of-the-art generative protein co-design model with the flexibility of multimodal protein generation.

2. Background

We aim to model discrete data where a sequence $x \in \{1, \dots, S\}^D$ has D dimensions, each taking on one of S states. For ease of exposition, we will assume $D = 1$; all results hold for $D > 1$ as discussed in App. E. We first explain a class of continuous time discrete stochastic processes called Continuous Time Markov Chains (CTMCs) (Norris, 1998) and then describe the link to probability flows.

2.1. Continuous Time Markov Chains.

A sequence trajectory x_t over time $t \in [0, 1]$ that follows a CTMC alternates between resting in its current state and periodically jumping to another randomly chosen state. We show example trajectories in Fig. 1B. The frequency and destination of the jumps are determined by the rate matrix $R_t \in \mathbb{R}^{S \times S}$ with the constraint its off-diagonal elements are non-negative. The probability x_t will jump to a different state j is $R_t(x_t, j)\mathrm{d}t$ for the next infinitesimal time step $\mathrm{d}t$. We can write the transition probability as

$$p_{t+\mathrm{d}t|t}(j|x_t) = \begin{cases} R_t(x_t, j)\mathrm{d}t & \text{for } j \neq x_t \\ 1 + R_t(x_t, x_t)\mathrm{d}t & \text{for } j = x_t \end{cases} \quad (1)$$

$$= \delta\{x_t, j\} + R_t(x_t, j)\mathrm{d}t \quad (2)$$

where $\delta\{i, j\}$ is the Kronecker delta which is 1 when $i = j$ and is otherwise 0 and $R_t(x_t, x_t) := -\sum_{k \neq x} R_t(x_t, k)$ in order for $p_{t+\mathrm{d}t|t}(\cdot|i)$ to sum to 1. We use compact notation Eq. (2) in place of Eq. (1). Therefore, $p_{t+\mathrm{d}t|t}$ is a Categorical distribution with probabilities $\delta\{x_t, \cdot\} + R_t(x_t, \cdot)\mathrm{d}t$ that we denote as $\mathrm{Cat}(\delta\{x_t, j\} + R_t(x_t, j)\mathrm{d}t)$:

$$j \sim p_{t+\mathrm{d}t|t}(j|x_t) \iff j \sim \mathrm{Cat}(\delta\{x_t, j\} + R_t(x_t, j)\mathrm{d}t).$$

In practice, we need to simulate the sequence trajectory with finite time intervals Δt. A sequence trajectory can be simulated with Euler steps (Sun et al., 2023b)

$$x_{t+\Delta t} \sim \mathrm{Cat}(\delta\{x_t, x_{t+\Delta t}\} + R_t(x_t, x_{t+\Delta t})\Delta t), \quad (3)$$

where the sequence starts from an initial sample $x_0 \sim p_0$ at time $t = 0$. The rate matrix R_t along with an initial distribution p_0 together define the CTMC.

2.2. Kolmogorov equation

For a sequence trajectory following the dynamics of a CTMC, we write its marginal distribution at time t as $p_t(x_t)$. The Kolmogorov equation allows us to relate the rate matrix R_t to the change in $p_t(x_t)$. It has the form:

$$\partial_t p_t(x_t) = \underbrace{\sum_{j \neq x_t} R_t(j, x_t) p_t(j)}_{\text{incoming}} - \underbrace{\sum_{j \neq x_t} R_t(x_t, j) p_t(x_t)}_{\text{outgoing}} \quad (4)$$

The difference between the incoming and outgoing probability mass is the time derivative of the marginal $\partial_t p_t(x_t)$. Using our definition of $R_t(x_t, x_t)$, Eq. (4) can be succinctly written as $\partial_t p_t = R_t^\top p_t$ where the marginals are treated as probability mass vectors: $p_t \in [0, 1]^S$. This defines an Ordinary Differential Equation (ODE) in a vector space. We refer to the series of distributions $p_t \; \forall t \in [0, 1]$ satisfying the ODE as a *probability flow*.

Key terms: A **CTMC** is defined by an initial distribution p_0 and rate matrix R_t. Samples along CTMC dynamics are called a **sequence trajectory** x_t. The **probability flow** p_t is the marginal distribution of x_t at every time t. We say R_t **generates** p_t if $\partial_t p_t = R_t^\top p_t \; \forall t \in [0, 1]$.

3. Discrete Flow Models

A Discrete Flow Model (DFM) is a **Discrete** data generative model built around a probability **Flow** that interpolates from noise to data. To sample new datapoints, we simulate a sequence trajectory that matches the noise to data probability flow. The flow construction allows us to combine DFM with continuous data flow models to define a multimodal generative model. Proofs for all propositions are in App. B.

3.1. A Flow Model for Sampling Discrete Data

We start by constructing the data generating probability flow referred to as the *generative flow*, p_t, that we will later sample from using a CTMC. The generative flow interpolates from noise to data where $p_0(x_0) = p_{\text{noise}}(x_0)$ and $p_1(x_1) = p_{\text{data}}(x_1)$. Since p_t is complex to consider directly, the insight of flow matching is to define p_t using a simpler datapoint conditional flow, $p_{t|1}(\cdot|x_1)$ that we will be able to write down explicitly. We can then define p_t as

$$p_t(x_t) := \mathbb{E}_{p_{\text{data}}(x_1)} \left[p_{t|1}(x_t|x_1) \right]. \quad (5)$$

The conditional flow, $p_{t|1}(\cdot|x_1)$ interpolates from noise to the datapoint x_1. The conditioning allows us to write the flow down in closed form. We are free to define $p_{t|1}(\cdot|x_1)$ as needed for the specific application. The conditional flows we use in this paper linearly interpolate towards x_1 from a uniform prior or an artificially introduced mask state, M:

$$p^{\text{unif}}_{t|1}(x_t|x_1) = \text{Cat}(t\delta\{x_1, x_t\} + (1-t)\tfrac{1}{S}), \quad (6)$$
$$p^{\text{mask}}_{t|1}(x_t|x_1) = \text{Cat}(t\delta\{x_1, x_t\} + (1-t)\delta\{M, x_t\}).$$

We require our conditional flow to converge on the datapoint x_1 at $t = 1$, i.e. $p_{t|1}(x_t|x_1) = \delta\{x_1, x_t\}$. We also require that the conditional flow starts from noise at $t = 0$, i.e. $p_{t|1}(x_t|x_1) = p_{\text{noise}}(x_t)$. In our examples, $p^{\text{unif}}_{\text{noise}}(x_t) = \frac{1}{S}$ and $p^{\text{mask}}_{\text{noise}}(x_t) = \delta\{M, x_t\}$. These two requirements ensure our generative flow, p_t, defined in Eq. (5) interpolates from p_{noise} at $t = 0$ towards p_{data} at $t = 1$ as desired. Next, we will show how to sample from the generative flow by exploiting p_t's decomposition into conditional flows.

3.1.1. Sampling

To sample from p_{data} using the generative flow, p_t, we need access to a rate matrix $R_t(x_t, j)$ that generates p_t. Given a $R_t(x_t, j)$, we could use Eq. (3) to simulate a sequence trajectory that begins with marginal distribution p_{noise} at $t = 0$ and ends with marginal distribution p_{data} at $t = 1$. The definition of p_t in Eq. (5) suggests $R_t(x_t, j)$ can also be derived as an expectation over a simpler conditional rate matrix. Define $R_t(x_t, j|x_1)$ as a datapoint conditional rate matrix that generates $p_{t|1}(x_t|x_1)$. We now show $R_t(x_t, j)$ can indeed be defined as an expectation over $R_t(x_t, j|x_1)$.

Proposition 3.1. *If $R_t(x_t, j|x_1)$ is a rate matrix that generates the conditional flow $p_{t|1}(x_t|x_1)$, then*

$$R_t(x_t, j) := \mathbb{E}_{p_{1|t}(x_1|x_t)} \left[R_t(x_t, j|x_1) \right] \quad (7)$$

is a rate matrix that generates p_t defined in Eq. (5). The expectation is taken over $p_{1|t}(x_1|x_t) = \frac{p_{t|1}(x_t|x_1) p_{\text{data}}(x_1)}{p_t(x_t)}$.

Our aim now is to calculate $R_t(x_t, j|x_1)$ and $p_{1|t}(x_1|x_t)$ to plug into Eq. (7). $p_{1|t}(x_1|x_t)$ is the distribution predicting clean data x_1 from noisy data x_t and in Sec. 3.1.2, we will train a neural network $p^\theta_{1|t}(x_1|x_t)$ to approximate it. In Sec. 3.2, we will show how to derive $R_t(x_t, j|x_1)$ in closed form. Sampling pseudo-code is provided in Alg. 1.

Algorithm 1 DFM Sampling

init $t = 0, x_0 \sim p_0$, choice of $R_t(x_t, \cdot|x_1)$ (Sec. 3.2)
while $t < 1$ **do**
  $R^\theta_t(x_t, \cdot) \leftarrow \mathbb{E}_{p^\theta_{1|t}(x_1|x_t)} \left[R_t(x_t, \cdot|x_1) \right]$
  $x_{t+\Delta t} \sim \text{Cat}\left(\delta\{x_t, x_{t+\Delta t}\} + R^\theta_t(x_t, x_{t+\Delta t}) \Delta t \right)$
  $t \leftarrow t + \Delta t$
end while
return x_1

We discuss further CTMC sampling methods in App. G. Our construction of the generative flow from conditional flows

Table 1. Comparison between continuous space linear interpolant flow models and DFMs with masking. Both start with a conditional flow $p_{t|1}(x_t|x_1)$ interpolating between data and noise. For continuous, $p_{t|1}(x_t|x_1) = \mathcal{N}(tx_1, (1-t)^2 I)$ and for discrete we use $p^{\text{mask}}_{t|1}$. Solving the Fokker-Planck or Kolmogorov equations with $p_{t|1}(x_t|x_1)$ gives a data conditioned process, specified either by the velocity field (ν_t) or the rate matrix (R_t). We train a model to learn the unconditional process – written analytically as the expected value of the conditional quantity – which is then used for sampling. The side-by-side comparison reveals the similar forms of each quantity.

QUANTITY	CONTINUOUS	DISCRETE
FOKKER-PLANCK-KOLMOGOROV	$\partial_t p_t = -\nabla \cdot (v_t p_t)$	$\partial_t p_t = R_t^\top p_t$
CONDITIONAL PROCESS	$\nu_t(x_t\|x_1) = \frac{x_1 - x_t}{1-t}$	$R_t(x_t, j\|x_1) = \frac{\delta\{j,x_1\}}{1-t}\delta\{x_t, M\}$
GENERATIVE PROCESS	$\nu_t(x_t) = \mathbb{E}_{p_{1\|t}(x_1\|x_t)}[\nu_t(x_t\|x_1)]$	$R_t(x_t, j) = \mathbb{E}_{p_{1\|t}(x_1\|x_t)}[R(x_t, j\|x_1)]$
GENERATIVE SAMPLING	$x_{t+\Delta t} = x_t + v_t(x_t)\Delta t$	$x_{t+\Delta t} \sim \text{Cat}(\delta\{x_t, x_{t+\Delta t}\} + R_t(x_t, x_{t+\Delta t})\Delta t)$

is analogous to the construction of generative probability paths from conditional probability paths in Lipman et al. (2023), where instead of a continuous vector field generating the probability path, we have a rate matrix generating the probability flow. We expand on these links in Table. 1.

3.1.2. TRAINING

We train a neural network with parameters θ, $p^\theta_{1|t}(x_t|x_1)$, to approximate the true denoising distribution using the standard cross-entropy i.e. learning to predict the clean datapoint x_1 when given noisy data $x_t \sim p_{t|1}(x_t|x_1)$.

$$\mathcal{L}_{\text{ce}} = \mathbb{E}_{p_{\text{data}}(x_1)\mathcal{U}(t;0,1)p_{t|1}(x_t|x_1)}\left[\log p^\theta_{1|t}(x_1|x_t)\right] \quad (8)$$

where $\mathcal{U}(t;0,1)$ is a uniform distribution on $[0,1]$. x_t can be sampled from $p_{t|1}(x_t|x_1)$ in a simulation-free manner by using the explicit form we wrote down for $p_{t|1}$ e.g. Eq. (6). In App. C, we analyse how $\mathcal{L}_{\text{ce}}$ relates to the model log-likelihood and its relation to the Evidence Lower Bound (ELBO) used to train diffusion models. We stress that $\mathcal{L}_{\text{ce}}$ does not depend on $R_t(x_t, j|x_1)$ and so we can postpone the choice of $R_t(x_t, j|x_1)$ until after training. This enables inference time flexibility in how our discrete data is sampled.

3.2. Choice of Rate Matrix

The missing piece in Eq. (7) is a conditional rate matrix $R_t(x_t, j|x_1)$ that generates the conditional flow $p_{t|1}(x_t|x_1)$. There are many choices for $R_t(x_t, j|x_1)$ that all generate the same $p_{t|1}(x_t|x_1)$ as we later show in Prop. 3.3. In order to proceed, we start by giving one valid choice of rate matrix and from this, build a set of rate matrices that all generate $p_{t|1}$. At inference time, we can then pick the rate matrix from this set that performs the best. Our starting choice for a rate matrix that generates $p_{t|1}$ is defined for $x_t \neq j$ as,

$$R^*_t(x_t, j|x_1) := \frac{\text{ReLU}\left(\partial_t p_{t|1}(j|x_1) - \partial_t p_{t|1}(x_t|x_1)\right)}{S \cdot p_{t|1}(x_t|x_1)}$$

where $\text{ReLU}(a) = \max(a, 0)$ and $\partial_t p_{t|1}$ can be found by differentiating our explicit form for $p_{t|1}$. This assumes $p_{t|1}(x_t|x_1) > 0$, see App. B.2 for the full form. We first heuristically justify R^*_t and then prove it generates $p_{t|1}(x_t|x_1)$ in Prop. 3.2. R^*_t can be understood as distributing probability mass to states that require it. If $\partial_t p_{t|1}(j|x_1) > \partial_t p_{t|1}(x_t|x_1)$ then state j needs to gain more probability mass than the current state x_t resulting in a positive rate. If $\partial_t p_{t|1}(j|x_1) \leq \partial_t p_{t|1}(i|x_1)$ then state x_t should give no mass to state j hence the ReLU. This rate should then be normalized by the probability mass in the current state. The ReLU ensures off-diagonal elements of R^*_t are positive and is inspired by Zhang et al. (2023).

Proposition 3.2. *Assuming zero mass states, $p_{t|1}(j|x_1) = 0$, have $\partial_t p_{t|1}(j|x_1) = 0$, then R^*_t generates $p_{t|1}(x_t|x_1)$.*

The proof is easy to derive by substituting R^*_t along with $p_{t|1}(x_t|x_1)$ into the Kolmogorov equation Eq. (4). The forms for $R^*_t(x_t, j|x_1)$ under $p^{\text{unif}}_{t|1}$ or $p^{\text{mask}}_{t|1}$ are simple

$$R^{*\text{unif}}_t = \frac{\delta\{x_1,j\}(1-\delta\{x_1,x_t\})}{1-t}, \quad R^{*\text{mask}}_t = \frac{\delta\{x_1,j\}\delta\{x_t,M\}}{1-t}$$

as we derive in App. F. Using R^*_t as a starting point, we now build out a set of rate matrices that all generate $p_{t|1}$. We can accomplish this by adding on a second rate matrix that is in detailed balance with $p_{t|1}$.

Proposition 3.3. *Let R^{DB}_t be a rate matrix that satisfies the detailed balance condition for $p_{t|1}$,*

$$p_{t|1}(i|x_1)R^{\text{DB}}_t(i, j|x_1) = p_{t|1}(j|x_1)R^{\text{DB}}_t(j, i|x_1), . \quad (9)$$

Let R^η_t be defined by R^*_t, R^{DB}_t and parameter $\eta \in \mathbb{R}^{\geq 0}$,

$$R^\eta_t := R^*_t + \eta R^{\text{DB}}_t.$$

Then we have R^η_t generates $p_{t|1}(x_t|x_1)$, $\forall \eta \in \mathbb{R}^{\geq 0}$.

The detailed balance condition intuitively enforces the incoming probability mass, $p_{t|1}(j|x_1)R^{\text{DB}}_t(j, i|x_1)$ to equal the outgoing probability mass, $p_{t|1}(i|x_1)R^{\text{DB}}_t(i, j|x_1)$. Therefore, R^{DB}_t has no overall effect on the probability flow and can be added on to R^*_t with the combined rate still generating $p_{t|1}$. In many cases, Eq. (9) is easy to solve for R^{DB}_t due to the explicit relation between elements of R^{DB}_t as we exemplify in App. F. Detailed balance has been used previously in CTMC generative models (Campbell et al., 2022) to make post-hoc inference adjustments.

CTMC stochasticity. We now have a set of rate matrices, $\{R_t^\eta : \eta \geq 0\}$, that all generate $p_{t|1}$. We can plug any one of these into our definition for $R_t(x_t, j)$ (Eq. (7)) and sample novel datapoints using Alg. 1. The chosen value for η will influence the dynamics of the CTMC we are simulating. For large values of η, the increased influence of R_t^{DB} will cause large exchanges of probability mass between states. This manifests as increasing the frequency of jumps occurring in the sequence trajectory. This leads to a short auto-correlation time for the CTMC and a high level of unpredictability of future states given the current state. We refer to the behaviour that η controls as *CTMC stochasticity*. Fig. 1B shows examples of high and low η.

On a given task, we expect there to be an optimal stochasticity level. Additional stochasticity improves performance in continuous diffusion models (Cao et al., 2023; Xu et al., 2023), but too much stochasticity can result in a poorly performing degenerate CTMC. In some cases, setting $\eta = 0$, i.e. using R_t^*, results in the minimum possible number of jumps because the ReLU within R_t^* removes state pairs that needlessly exchange mass (Zhang et al., 2023).

Proposition 3.4. *For $p_{t|1}^{\text{unif}}$ and $p_{t|1}^{\text{mask}}$, R_t^* generates $p_{t|1}$ whilst minimizing the expected number of jumps during the sequence trajectory. This assumes multi-dimensional data under the factorization assumptions listed in App. E.*

3.3. DFMs Recipe

We now summarize the key steps of a DFM. PyTorch code for a minimal DFM implementaton is provided in App. F.

1. Define the desired noise schedule $p_{t|1}(x_t|x_1)$ (Sec. 3.1).
2. Train denoising model $p_{1|t}^\theta(x_1|x_t)$ (Sec. 3.1.2).
3. Choose rate matrix R_t^η (Sec. 3.2).
4. Run sampling (Alg. 1).

4. Multimodal Protein Generative Model

Using our flow formulation on discrete state spaces, we can now combine a DFM with a flow on a continuous space to define a multimodal generative flow. We use this to perform protein joint structure-sequence generation. A protein can be modeled as a linear chain of residues, each with an assigned amino acid and 3D atomic coordinates. Protein co-design aims to jointly generate the amino acids (sequence) and coordinates (structure). Prior works have used a generative model on one modality (sequence or structure) with a separate model to predict the other (see Sec. 5). Instead, our approach uses a single generative model to jointly sample both modalities: a DFM for the sequence and a flow model, FrameFlow (Yim et al., 2023a), for the structure. We refer to our multimodal flow model as **Multiflow**.

4.1. Multimodal Flows

Following FrameFlow, we refer to the protein structure as the *backbone* atomic coordinates of each residue. We leave modeling side-chain atoms as a follow-up work. The structure is represented as elements of $\text{SE}(3)$ to capture the rigidity of the local frames along the backbone (Yim et al., 2023b). A protein of length D residues can then be represented as $\{(x^d, r^d, a^d)\}_{d=1}^D$ where $x \in \mathbb{R}^3$ is the translation of the residue's Carbon-α atom, $r \in \text{SO}(3)$ is a rotation matrix of the residue's local frame with respect to global reference frame, and $a \in \{1, \dots, 20\} \cup \{M\}$ is one of 20 amino acids or the mask state M. For brevity, we refer to the residue state as $T^d = (x^d, r^d, a^d)$ and let the full protein's structure and sequence as $\mathbf{T} = \{T^d\}_{d=1}^D$. We define the multimodal conditional flow as $p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)$ which is a shorthand for a probability density over the continuous variables and a probability mass function over the discrete variables. We define $p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)$ to factorize over both dimensions and modality.

$$p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) := \prod_{d=1}^D p_{t|1}(x_t^d|x_1^d)p_{t|1}(r_t^d|r_1^d)p_{t|1}(a_t^d|a_1^d) \quad (10)$$

Following Yim et al. (2023a), $p_{t|1}(x_t^d|x_1^d)$ and $p_{t|1}(r_t^d|r_1^d)$ are defined implicitly through specifying how samples x_t^d, r_t^d are generated from $p_{t|1}(x_t^d|x_1^d)$, $p_{t|1}(r_t^d|r_1^d)$,

$$x_t^d = tx_1^d + (1-t)x_0^d,\ x_0^d \sim \mathcal{N}(0, I) \quad (11)$$

$$r_t^d = \exp_{r_0^d}\left(t\log_{r_0^d}(r_1^d)\right),\ r_0^d \sim \mathcal{U}_{\text{SO}(3)}, \quad (12)$$

where $\exp$ and $\log$ are the exponential and logarithmic maps. $\mathcal{U}_{\text{SO}(3)}$ is the uniform distribution on $\text{SO}(3)$. Following Sec. 3, $p_{t|1}(a_t^d|a_1^d)$ is defined explicitly.

$$p_{t|1}(a_t^d|a_1^d) = \text{Cat}\big(t\delta\left\{a_1^d, a_t^d\right\} + (1-t)\delta\left\{M, a_t^d\right\}\big) \quad (13)$$

Our conditional trajectory that follows this conditional flow will be an ODE on the continuous modalities with a CTMC for the amino acids. The conditional ODE on translations and rotations is parameterized through conditional velocities $v_x^d(x_t^d|x_1^d) \in \mathbb{R}^3$, $v_r^d(r_t^d|r_1^d) \in \text{Tan}_{r_t^d}\text{SO}(3)$ (Yim et al., 2023a). v_x^d is a standard Euclidean vector field whereas v_r^d is a vector field on the Riemannian Manifold $\text{SO}(3)$ (Chen & Lipman, 2023). The trajectory can be simulated using Euler steps with step size Δt,

$$\begin{aligned} x_{t+\Delta t}^d &= x_t^d + v_x^d(x_t^d|x_1^d)\Delta t \\ r_{t+\Delta t}^d &= \exp_{r_t^d}(\Delta t \cdot v_r^d(r_t^d|r_1^d)) \\ a_{t+\Delta t}^d &\sim \text{Cat}\big(\delta\left\{a_t^d, a_{t+\Delta t}^d\right\} + R_t^d(a_t^d, a_{t+\Delta t}^d|a_1^d)\Delta t\big). \end{aligned} \quad (14)$$

We choose v_x^d such that it individually generates the $p_{t|1}(x_t^d|x_1^d)$ given by Eq. (11) if it were simulated by itself in $\mathbb{R}^3$. Similarly, for v_r^d and R_t^d, they are chosen such

that they individually generate $p_{t|1}(r_t^d|r_1^d)$ (Eq. (12)) and $p_{t|1}(a_t^d|a_1^d)$ (Eq. (13)) respectively. The explicit forms for v_x^d, v_r^d and R_t^d are as follows,

$$\begin{aligned} v_x^d(x_t^d|x_1^d) &= (x_1^d - x_t^d)/(1-t) \\ v_r^d(r_t^d|r_1^d) &= \log_{r_t^d}(r_1^d)/(1-t) \\ R_t^d(a_t^d, j^d|a_1^d) &= \delta\left\{j^d, a_1^d\right\}\delta\left\{a_t^d, M\right\}/(1-t). \end{aligned} \tag{15}$$

wth velocities following Yim et al. (2023a) and rate matrix derived in App. F.1 assuming $\eta = 0$. The following proposition verifies these choices are consistent with our initial definition of $p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)$.

Proposition 4.1. *The multimodal process defined by Eq. (15) has the flow $p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)$ given by Eq. (10).*

We would now like to be able to sample a trajectory that follows the unconditional flow. Mirroring Prop. 3.1, we again find that the desired unconditional velocities and rate matrix are expectations of their respective conditional quantities.

Proposition 4.2. *The following velocities and rate matrix together generate $p_t(\mathbf{T}_t) = \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right]$,*

$$\begin{aligned} v_x^d(\mathbf{T}_t) &= \mathbb{E}_{p_{1|t}(x_1^d|\mathbf{T}_t)}\left[v_x^d(x_t^d|x_1^d)\right] \\ v_r^d(\mathbf{T}_t) &= \mathbb{E}_{p_{1|t}(r_1^d|\mathbf{T}_t)}\left[v_r^d(r_t^d|r_1^d)\right] \\ R_t^d(\mathbf{T}_t, j^d) &= \mathbb{E}_{p_{1|t}(a_1^d|\mathbf{T}_t)}\left[R_t^d(a_t^d, j^d|a_1^d)\right]. \end{aligned}$$

We note that even though the conditional flow is defined to factorize over modality and dimension, the unconditional generative flow has coupled modalities and dimensions because each velocity and rate matrix depends on the entire corrupted protein state $\mathbf{T}_t$.

Thus far, we have assumed the same noise level in all modalities. To enable flexible sampling options, we can use a noise level for the structure, t, that is independent to the noise level of the sequence, $\tilde{t}$ (Albergo et al., 2023). We let $\mathbf{T}_{t,\tilde{t}} = (x_t^{1:D}, r_t^{1:D}, a_{\tilde{t}}^{1:D})$ and use a conditional flow of

$$p_{t,\tilde{t}|1}(\mathbf{T}_{t,\tilde{t}}|\mathbf{T}_1) = \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d)p_{t|1}(r_t^d|r_1^d)p_{\tilde{t}|1}(a_{\tilde{t}}^d|a_1^d).$$

The unconditional flow then becomes $p_{t,\tilde{t}}(\mathbf{T}_{t,\tilde{t}}) = \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}[p_{t,\tilde{t}|1}(\mathbf{T}_{t,\tilde{t}}|\mathbf{T}_1)]$, with the expectations for the unconditional velocities and rate matrix in Prop. 4.2 now computed using $p_{1|t,\tilde{t}}(\cdot|\mathbf{T}_{t,\tilde{t}})$ instead of $p_{1|t}(\cdot|\mathbf{T}_t)$.

4.2. Training

During training, our network will take as input the noised protein $\mathbf{T}_{t,\tilde{t}}$ and predict the denoised translations $\hat{x}_1(\mathbf{T}_{t,\tilde{t}})$, rotations $\hat{r}_1(\mathbf{T}_{t,\tilde{t}})$, and amino acid distribution $p_\theta(a_1|\mathbf{T}_{t,\tilde{t}})$. We then parameterize the unconditional velocities and rate matrix in terms of these predicted quantities.

$$\begin{aligned} \nu_x^d(\mathbf{T}_{t,\tilde{t}}) &= \tfrac{\hat{x}_1^d(\mathbf{T}_{t,\tilde{t}}) - x_t^d}{1-t},\ \nu_r^d(\mathbf{T}_{t,\tilde{t}}) = \tfrac{\log_{r_t^d}(\hat{r}_1^d(\mathbf{T}_{t,\tilde{t}}))}{1-t}, \\ R_{\tilde{t}}^{\theta d}(\mathbf{T}_{t,\tilde{t}}, j) &= \tfrac{p_\theta(a_1^d = j|\mathbf{T}_{t,\tilde{t}})}{1-\tilde{t}}\delta\left\{a_{\tilde{t}}^d, M\right\}. \end{aligned}$$

In order for these to match their optimum values given in Prop. 4.2, we minimize the following loss

$$\begin{aligned} \mathbb{E}\Big[\textstyle\sum_{d=1}^{D} \tfrac{\left|\left|\hat{x}_1^d(\mathbf{T}_{t,\tilde{t}}) - x_1^d\right|\right|^2}{1-t} - \log p_\theta(a_1^d|\mathbf{T}_{t,\tilde{t}}) \\ + \tfrac{\left|\left|\log_{r_t^d}\left(\hat{r}_1^d(\mathbf{T}_t)\right) - \log_{r_t^d}\left(r_1^d\right)\right|\right|^2}{1-t}\Big]. \end{aligned} \tag{16}$$

where the expectation is over $t, \tilde{t} \sim \mathcal{U}(0,1)$, $\mathbf{T}_{1,1} \sim p_{\text{data}}$ and $\mathbf{T}_{t,\tilde{t}} \sim p_{t,\tilde{t}|1}(\mathbf{T}_{t,\tilde{t}}|\mathbf{T}_{1,1})$. Our independent $t, \tilde{t}$ objective enables the model to learn over different relative levels of corruption between the sequence and structure. Eq. (16) corresponds to the flow matching loss for continuous data and the DFMs loss Eq. (8) for discrete amino acids. The neural network architecture is modified from FrameFlow with a larger transformer, smaller Invariant Point Attention, and extra multi-layer perception head to predict the amino acid logits.

4.3. Sampling

To sample the generative model, we use the update equations from Eq. (14) but with the learned unconditional velocities and rate matrix. Furthermore, we find sample quality can be improved by using the exponential rate scheduler for rotations from Bose et al. (2023). In practice, this means v_r^d has the following form,

$$\nu_r^d(\mathbf{T}_{t,\tilde{t}}) = c \cdot \log_{r_t^d}(\hat{r}_1^d(\mathbf{T}_{t,\tilde{t}})).$$

We use $c = 10$ following (Yim et al., 2023a). When sampling the amino acids, we also found it beneficial to utilize purity (Tang et al., 2022) to choose which indices to unmask at each step. The advantage of training with decoupled time schedules is that we have freedom to arbitrarily sample with any combination of $(t, \tilde{t})$. We use this to perform conditional inpainting where one of the modalities is fixed by setting t or $\tilde{t}$ equal to 1. For example, setting $t = 1$ then using Euler steps to update $\tilde{t}$ from $0 \to 1$ performs sequence generation conditioned on the structure. We summarize the capabilities in Fig. 1C and in Table. 2.

Table 2. Flexible multimodal sampling.

	Codesign	Inverse folding	Forward folding
x_t, r_t	$t: 0 \to 1$	$t = 1$	$t: 0 \to 1$
$a_{\tilde{t}}$	$\tilde{t}: 0 \to 1$	$\tilde{t}: 0 \to 1$	$\tilde{t} = 1$

5. Related Work

Discrete Diffusion Models. Our continuous time flow builds on work that extends discrete diffusion (Hoogeboom

et al., 2021; Austin et al., 2021) to continuous time (Campbell et al., 2022; Sun et al., 2023b; Santos et al., 2023; Lou et al., 2023) but we simplify and extend the framework. We are not restricted to noising processes that can be defined by a matrix exponential as we just write $p_{t|1}$ down directly and we have the freedom to choose $R_t(x_t, j|x_1)$ at inference time rather than being restricted to the time reversal. We show how DFMs encompasses prior discrete diffusion models in App. H. For molecular retrosynthesis, Igashov et al. (2023) also considered a data conditional process, but did not build a modeling framework around it. Zhang et al. (2023) constructed low-stochasticity rate matrices and their derivation provides the building blocks of Prop. 3.2. Some works have built a multimodal diffusion model for molecule generation (Peng et al., 2023; Vignac et al., 2023b; Hua et al., 2023) whereas we focus on protein co-design using flows. We discuss further related work in App. D.

Protein Generation. Diffusion and flow models have risen in popularity for generating novel and diverse protein backbones (Yim et al., 2023b;a; Bose et al., 2023; Lin & AlQuraishi, 2023; Ingraham et al., 2023). RFDiffusion achieved notable success by generating proteins validated in wet-lab experiments (Watson et al., 2023). However, these methods required a separate model for sequence generation. Some works have focused only on sequence generation with diffusion models (Alamdari et al., 2023; Gruver et al., 2023; Yang et al., 2023; Yi et al., 2023). We focus on co-design which aims to jointly generate the structure and sequence.

Prior works have attempted co-design. ProteinGenerator (Lisanza et al., 2023) performs Euclidean diffusion over one-hot amino acids while predicting the structure at each step with RosettaFold (Baek et al., 2021). Conversely, Protpardelle (Chu et al., 2023) performs Euclidean diffusion over structure while iteratively predicting the sequence. Multiflow instead uses a generative model over *both* the structure and sequence which allows for flexibility in conditioning at inference time (see Sec. 6.2.1). Luo et al. (2022); Shi et al. (2022) are co-design methods, but are limited to generating CDR loops on antibodies. Lastly, Anand & Achim (2022) presented diffusion on structure and sequence, but did not report standard evaluation metrics nor is code available.

6. Experiments

We first show that tuning stochasticity at sample time improves pure discrete generative modeling performance by modeling text data. We then evaluate Multiflow, the first flow model on discrete and continuous state spaces. We show Multiflow provides state-of-the-art-performance on protein generation compared to prior approaches that do not generate using a true multimodal generative model. Finally, we investigate Multiflow's crossmodal properties of how varying the sequence sampling affects the structure.

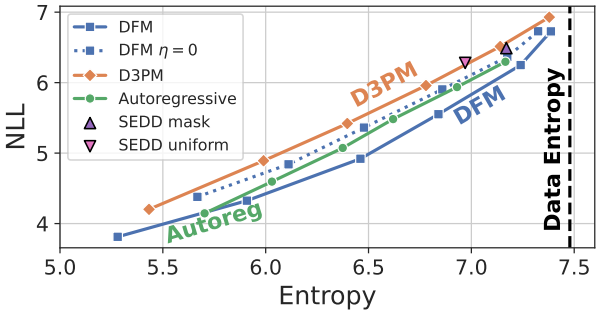


Figure 2. Negative log-likelihood as measured by GPT-J-6B versus sample entropy. For DFM, D3PM and autoregressive, we sweep the logit temperature for $p^{\theta}_{1|t}(x_1|x_t)$ over $\{0.5, 0.6, , \ldots, 1\}$. We aim to minimize NLL whilst staying close to the dataset entropy.

6.1. Text Modeling

Set-up. We model the text dataset, text8 (Mahoney, 2006), which is 100MB of text from English Wikipedia. We model at the character level, following (Austin et al., 2021), with $S = 28$ categories for 26 lowercase letters, a white-space and a mask token. We split the text into chunks of length $D = 256$. We train a DFM using $p^{\text{mask}}_{t|1}$ and parameterize the denoising network using a transformer with 86M non-embedding parameters, full details are in App. I.

Results. Text samples are evaluated following Strudel et al. (2022). A much larger text model, we use GPT-J-6B (Wang & Komatsuzaki, 2021), is used to evaluate the negative log-likelihood (NLL) of the generated samples. The NLL metric alone can be gamed by repeating similar sequences, so the token distribution entropy is also measured. Good samples should have both low NLL and entropy close to the data distribution. For a given value of η, we create a Pareto-frontier in NLL vs entropy space by varying the temperature applied to the $p^{\theta}_{1|t}(x_1|x_t)$ logits during the softmax operation. Fig. 2 plots the results for varying levels of η and sampling temperature. For comparison, we also include results for the discrete diffusion D3PM method with absorbing state corruption (Austin et al., 2021) as well as the Score Entropy Discrete Diffusion (SEDD) method of Lou et al. (2023) using both uniform and absorbing style corruption. SEDD does not have logits that can be temperature scaled and so only single points in NLL vs entropy space are shown. We find the DFM performs better than D3PM and SEDD due to our additional sample time flexibility. We are able to choose the value of η that optimizes the Pareto-frontier at sample time (here $\eta = 15$) whereas D3PM and SEDD do not have this flexibility. We show the full η sweep in App. I and show the frontier for $\eta = 0$ in Fig. 2. When $\eta = 0$, performance is similar to D3PM due to DFMs being a continuous time generalization of D3PM at this setting, see App. H.2. We also include results for an autoregressive model in Fig. 2 for reference; however, we note this is not a complete like-for-like comparison as autoregressive models require much less

compute to train than diffusion based models (Gulrajani & Hashimoto, 2023).

6.2. Protein generation

Metrics. Evaluating the quality of structure-sequence samples is performed with *self-consistency* which measures how consistent a generated sequence is with a generated structure by testing how accurately a protein folding network can predict the structure from the sequence. Specifically, either AlphaFold2 (Jumper et al., 2021) or ESMFold (Lin et al., 2023), is first used to predict a structure given only the generated sequence. Our results will use ESMFold but we show results with AlphaFold2 in App. J. Then, we calculate scRMSD: the Root Mean Squared Deviation between the generated and predicted structure's backbone atoms. The generated structure is called *designable* if scRMSD < 2Å.

Structure-only generative models such as RFdiffusion first use ProteinMPNN (PMPNN) (Dauparas et al., 2022) to predict a sequence given the generated structure in order to then be able to use the self-consistency metric. We present three variants of self-consistency:

- *Co-design 1*: use the sampled (structure, sequence) pair.
- *PMPNN 8*: take only the sampled structure and predict 8 sequences with PMPNN. Then use ESMFold to predict a new structure for each sequence. The final structure-sequence pair is the original sampled structure along with the PMPNN sequence with minimum scRMSD.
- *PMPNN 1*: same as PMPNN 8 except PMPNN only generates one sequence.

PMPNN 8 and PMPNN 1 evaluate only the quality of a model's generated structures whereas, for co-design models, Co-design 1 evaluates the quality of a model's generated (structure, sequence) pairs. The comparison between PMPNN 1 and Co-design 1 allows for evaluating the quality of co-designed sequences. PMPNN 8 is the procedure used in prior structure-only works. As our main metric of sample quality, we report *designability* as the percentage of designable samples. As a further sanity check, designable samples are then evaluated on *diversity* and *novelty*. We use FoldSeek (van Kempen et al., 2022) to report diversity as the number of unique clusters while novelty is the average TM-score (Zhang & Skolnick, 2005) of each sample to its most similar protein in PDB.

Training. Our training data consisted of length 60-384 proteins from the Protein Data Bank (PDB) (Berman et al., 2000) that were curated in Yim et al. (2023b) for a total of 18684 proteins. Training took 200 epochs over 3 days on 4 A6000 Nvidia GPUs using the AdamW optimizer (Loshchilov & Hutter, 2017) with learning rate 0.0001.

Distillation. Multiflow with PDB training generated highly designable structures. However, the co-designed sequences suffered from lower designability than PMPNN. Our analysis revealed the original PDB sequences achieved worse designability than PMPNN. We sought to improve performance by distilling knowledge from other models. To accomplish this, we first replaced the original sequence of each structure in the training dataset with the lowest scRMSD sequence out of 8 generated by PMPNN conditioned on the structure. Second, we generated synthetic structures of random lengths between 60-384 using an initial Multiflow model and added those that passed PMPNN 8 designability into the training dataset with the lowest scRMSD PMPNN sequence. We found that we needed to add only an extra 4179 examples to the original set of 18684 proteins to see a dramatic improvement. This procedure can be seen as a single step of reinforced self training (ReST) (Gulcehre et al., 2023).

6.2.1. Co-design results.

Following RFdiffusion's benchmark, we sample 100 proteins for each length 70, 100, 200, and 300. We sample Multiflow with 500 timesteps using a temperature of 0.1 (PMPNN also uses 0.1) and stochasticity level $\eta = 20$. We compare our structure quality to state-of-the-art structure generation method RFdiffusion. For co-design, we compare to Protpardelle and ProteinGenerator. All methods were ran using their publicly released code and evaluated identically.

Our results are presented in Table. 3 where report the average of three seeds for each metric – see Table. 6 for results with standard error. We find that Multiflow's co-design capabilities surpass previous co-design methods, none of which use a joint multimodal generation process. Multiflow generates sequences that are consistent with the generated structure at a comparable level to PMPNN which we see through comparing the Co-design 1 and PMPNN 1 designability. On pure structure generation, we find that Multiflow outperforms all baselines in terms of structure quality measured by PMPNN 8 designability. Multiflow also attains comparable diversity and novelty to previous approaches. We ablate our use of distillation and find that distillation results in overall designability improvements while also improving diversity. Finally, we train our exact same architecture except only modeling the structure on the distilled dataset using the loss presented in Yim et al. (2023a). We find our joint structure-sequence model achieves the same structural quality as the structure-only version, however, additionally including the *sequence* in our generative process induces extra *structural* diversity.

Crossmodal modulation. We next investigate how modulating the CTMC stochasticity of the sequence affects the structural properties of sampled proteins. Fig. 3 shows that varying the stochasticity level η results in a change of the secondary structure composition (Kabsch & Sander, 1983) of the sampled proteins. This is an example of the flexibil-

Table 3. Co-design results. Abbreviations: Designability (**DES.**), Diversity (**DIV.**), Novelty (**NOV.**). For Protpardelle, we report Co-design 1 as same numbers as PMPNN 1 since their co-design approach employs PMPNN. We note this is not co-generation since PMPNN is used while Multiflow explicitly learns co-generation without using PMPNN.

METHOD	CO-DESIGN 1			PMPNN 8			PMPNN 1		
	DES. (↑)	DIV. (↑)	NOV. (↓)	DES.	DIV.	NOV.	DES.	DIV.	NOV.
PROTPARDELLE	0.63*	38*	0.60*	0.90	47	**0.59**	0.63	38	**0.60**
PROTEINGENERATOR	0.37	35	0.69	0.89	75	0.65	0.78	64	0.66
RFDIFFUSION		N/A		0.87	**158**	0.63	0.66	111	0.64
MULTIFLOW	**0.86**	**143**	**0.61**	**0.99**	156	0.61	**0.88**	**143**	0.61
MULTIFLOW W/O DISTILLATION	0.42	72	0.62	0.89	126	0.62	0.71	101	0.63
MULTIFLOW W/O SEQUENCE		N/A		**0.99**	116	0.63	0.86	97	0.62

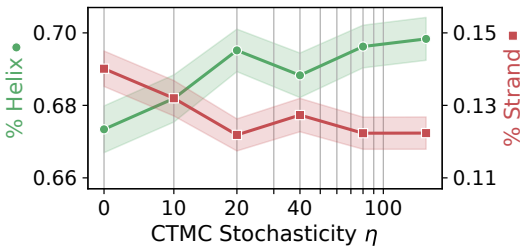


Figure 3. **Multiflow structural properties.** Average proportion of residues that are part of an alpha helix or beta strand versus the CTMC stochasticity level. Proportions of helices or strands can be desirable based on the family of proteins to generate (Vinothkumar & Henderson, 2010). Error bars show the standard error.

ity our multimodal framework provides to tune properties between data modalities at inference time.

6.2.2. FORWARD AND INVERSE FOLDING

Multiflow can achieve state-of-the-art codesign performance, but can accomplish more tasks as described in Fig. 1B and Table. 4. Expanding Multiflow to achieve competitive performance on all tasks is a future work. Here, we take the same model weights for co-design and evaluate forward and inverse folding *without additional training*. We compare performance to ESMFold and ProteinPMNN which are specialized models for forward and inverse folding. We curated a clustered test-out set of 449 monomeric proteins with length < 400 from the PDB using a date split of our training set. Details of forward/inverse folding and these experiments can be found in App. J. We find Multiflow can achieve very close performance with ProteinMPNN while it achieves poor results compared to ESMFold. This highlights a limitation that Multiflow cannot perform competitively at every generation task, but leaves exciting future work for a potential general-purpose generative model.

Table 4. Forward and inverse folding: mean ± std.

Method	Inverse folding scRMSD (↓)	Forward folding RMSD (↓)
ProteinMPNN	1.9 ± 2.7	N/A
ESMFold	N/A	2.7 ± 3.9
Multiflow	2.2 ± 2.6	15.3 ± 4.5

7. Discussion

We presented Discrete Flow Models (DFMs), a flow based generative model framework by making analogy to continuous state space flow models. Our formulation is simple to implement, removes limitations in defining corruption processes, and provides more sampling flexibility for improved performance compared to previous discrete diffusion models. Our framework enables easy application to multimodal generative problems which we apply to protein co-design. The combination of a DFM and FrameFlow enables state-of-the-art co-design with Multiflow. Future work includes to develop more domain specific models with DFMs and improve Multiflow's performance on all protein generation tasks including sidechain modeling.

Acknowledgments

The authors would like to thank Ricardo Baptista, Mathieu Le Provost, George Deligiannidis, Joe Benton, Bowen Jing, Hannes Stärk, Emile Mathieu, Luhuan Wu, Timur Garipov, Rachel Wu, Mingyu Choi, Sidney Lisanza, and Woody Ahern for helpful discussions. AC acknowledges support from the EPSRC CDT in Modern Statistics and Statistical Machine Learning (EP/S023151/1) JY was supported in part by an NSF-GRFP. JY, RB, and TJ acknowledge support from NSF Expeditions grant (award 1918839: Collaborative Research: Understanding the World Through Code), Machine Learning for Pharmaceutical Discovery and Synthesis (MLPDS) consortium, the Abdul Latif Jameel Clinic for Machine Learning in Health, the DTRA Discovery of Medical Countermeasures Against New and Emerging (DOMANE) threats program, the DARPA Accelerated Molecular Discovery program and the Sanofi Computational Antibody Design grant. IF is supported by the Office of Naval Research, the Howard Hughes Medical Institute (HHMI), and NIH (NIMH-MH129046). The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work. http://dx.doi.org/10.5281/zenodo.22558.

Impact statement

In this paper we work to advance general purpose generative modeling techniques, specifically those used for modeling discrete and multimodal data. We apply these techniques to the task of protein generation. Improving protein modeling capabilities can have wide ranging societal impacts and care must be taken to ensure these impacts are positive. For example, improved modeling capabilities can help design better enzymes and drug candidates that can then go on to improve the lives of many people. Conversely, these general purpose techniques could also be misused to design toxic substances. To mitigate these risks, we do not present any specific methods to apply Multiflow to tasks that could be easily adjusted to the design of harmful substances without expert knowledge.

References

Alamdari, S., Thakkar, N., van den Berg, R., Lu, A. X., Fusi, N., Amini, A. P., and Yang, K. K. Protein generation with evolutionary diffusion: sequence is all you need. *bioRxiv*, pp. 2023–09, 2023.

Albergo, M. S. and Vanden-Eijnden, E. Building normalizing flows with stochastic interpolants. *International Conference on Learning Representations*, 2023.

Albergo, M. S., Boffi, N. M., Lindsey, M., and Vanden-Eijnden, E. Multimarginal generative modeling with stochastic interpolants. *arXiv preprint arXiv:2310.03695*, 2023.

Anand, N. and Achim, T. Protein structure and sequence generation with equivariant denoising diffusion probabilistic models. *arXiv preprint arXiv:2205.15019*, 2022.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 2021.

Baek, M., DiMaio, F., Anishchenko, I., Dauparas, J., Ovchinnikov, S., Lee, G. R., Wang, J., Cong, Q., Kinch, L. N., Schaeffer, R. D., et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.

Bect, J. A unifying formulation of the fokker–planck–kolmogorov equation for general stochastic hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 2010.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E. J., Tiwari, M., and Bengio, E. Gflownet foundations. *Journal of Machine Learning Research*, 2023.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. The protein data bank. *Nucleic acids research*, 28(1): 235–242, 2000.

Bose, A. J., Akhound-Sadegh, T., Fatras, K., Huguet, G., Rector-Brooks, J., Liu, C.-H., Nica, A. C., Korablyov, M., Bronstein, M., and Tong, A. Se (3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.

Campbell, A., Benton, J., De Bortoli, V., Rainforth, T., Deligiannidis, G., and Doucet, A. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 2022.

Cao, Y., Chen, J., Luo, Y., and Zhou, X. Exploring the optimal choice for generative processes in diffusion models: Ordinary vs stochastic differential equations. *arXiv preprint arXiv:2306.02063*, 2023.

Chen, R. T. and Lipman, Y. Riemannian flow matching on general geometries. *arXiv preprint arXiv:2302.03660*, 2023.

Chen, T., Zhang, R., and Hinton, G. Analog bits: Generating discrete data using diffusion models with self-conditioning. *International Conference on Learning Representations*, 2023a.

Chen, Z., Yuan, H., Li, Y., Kou, Y., Zhang, J., and Gu, Q. Fast sampling via de-randomization for discrete diffusion models. *arXiv preprint arXiv:2312.09193*, 2023b.

Chow, S.-N., Huang, W., Li, Y., and Zhou, H. Fokker–planck equations for a free energy functional or markov process on a graph. *Archive for Rational Mechanics and Analysis*, 2012.

Chu, A. E., Cheng, L., El Nesr, G., Xu, M., and Huang, P.-S. An all-atom protein generative model. *bioRxiv*, pp. 2023–05, 2023.

Dauparas, J., Anishchenko, I., Bennett, N., Bai, H., Ragotte, R. J., Milles, L. F., Wicky, B. I., Courbet, A., de Haas, R. J., Bethel, N., et al. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378 (6615):49–56, 2022.

Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A. P., Caron, M., Geirhos, R., Alabdulmohsin, I., et al. Scaling vision transformers to 22 billion parameters. *International Conference on Machine Learning*, 2023.

Del Moral, P. and Penev, S. Stochastic processes: From applications to theory. *CRC Press*, 2017.

Dieleman, S., Sartran, L., Roshannai, A., Savinov, N., Ganin, Y., Richemond, P. H., Doucet, A., Strudel, R.,

Dyer, C., Durkan, C., et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

Floto, G., Jonsson, T., Nica, M., Sanner, S., and Zhu, E. Z. Diffusion on the probability simplex. *arXiv preprint arXiv:2309.02530*, 2023.

Gao, W., Mahajan, S. P., Sulam, J., and Gray, J. J. Deep learning in protein structural modeling and design. *Patterns*, 1(9), 2020.

Gillespie, D. T. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 2001.

Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. Diffuseq: Sequence to sequence text generation with diffusion models. *International Conference on Learning Representations*, 2023.

Graves, A., Srivastava, R. K., Atkinson, T., and Gomez, F. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023.

Gruver, N., Stanton, S., Frey, N. C., Rudner, T. G., Hotzel, I., Lafrance-Vanasse, J., Rajpal, A., Cho, K., and Wilson, A. G. Protein design with guided discrete diffusion. *arXiv preprint arXiv:2305.20009*, 2023.

Gulcehre, C., Paine, T. L., Srinivasan, S., Konyushkova, K., Weerts, L., Sharma, A., Siddhant, A., Ahern, A., Wang, M., Gu, C., et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Gulrajani, I. and Hashimoto, T. B. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 2023.

Han, X., Kumar, S., and Tsvetkov, Y. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. *arXiv preprint arXiv:2210.17432*, 2022.

Hastings, W. K. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 2020.

Hoogeboom, E., Nielsen, D., Jaini, P., Forré, P., and Welling, M. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 2021.

Hua, C., Luan, S., Xu, M., Ying, R., Fu, J., Ermon, S., and Precup, D. Mudiff: Unified diffusion for complete molecule generation. *arXiv preprint arXiv:2304.14621*, 2023.

Huang, C.-W., Lim, J. H., and Courville, A. C. A variational perspective on diffusion-based generative models and score matching. *Advances in Neural Information Processing Systems*, 2021.

Igashov, I., Schneuing, A., Segler, M., Bronstein, M., and Correia, B. Retrobridge: Modeling retrosynthesis with markov bridges. *arXiv preprint arXiv:2308.16212*, 2023.

Ingraham, J. B., Baranov, M., Costello, Z., Barber, K. W., Wang, W., Ismail, A., Frappier, V., Lord, D. M., Ng-Thow-Hing, C., Van Vlack, E. R., et al. Illuminating protein space with a programmable generative model. *Nature*, 2023.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 2021.

Kabsch, W. and Sander, C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules*, 22(12):2577–2637, 1983.

Karras, T., Aittala, M., Aila, T., and Laine, S. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35: 26565–26577, 2022.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kotelnikov, A., Baranchuk, D., Rubachev, I., and Babenko, A. Tabddpm: Modelling tabular data with diffusion models. *International Conference on Machine Learning*, 2023.

Li, X., Thickstun, J., Gulrajani, I., Liang, P. S., and Hashimoto, T. B. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 2022.

Lin, Y. and AlQuraishi, M. Generating novel, designable, and diverse protein structures by equivariantly diffusing oriented residue clouds. *arXiv preprint arXiv:2301.12485*, 2023.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637): 1123–1130, 2023.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling. *International Conference on Learning Representations*, 2023.

Lisanza, S. L., Gershon, J. M., Tipps, S. W. K., Arnoldt, L., Hendel, S., Sims, J. N., Li, X., and Baker, D. Joint generation of protein sequence and structure with rosettafold sequence space diffusion. *bioRxiv*, pp. 2023–05, 2023.

Liu, X., Gong, C., and Liu, Q. Flow straight and fast: Learning to generate and transfer data with rectified flow. *International Conference on Learning Representations*, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lou, A., Meng, C., and Ermon, S. Discrete diffusion language modeling by estimating the ratios of the data distribution. *Advances in Neural Information Processing Systems*, 2023.

Luo, S., Su, Y., Peng, X., Wang, S., Peng, J., and Ma, J. Antigen-specific antibody design and optimization with diffusion-based generative models for protein structures. *Advances in Neural Information Processing Systems*, 35: 9754–9767, 2022.

Ma, N., Goldstein, M., Albergo, M. S., Boffi, N. M., Vanden-Eijnden, E., and Xie, S. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

Mahoney, M. Large text compression benchmark. 2006. URL https://www.mattmahoney.net/dc/text.html.

Meng, C., Choi, K., Song, J., and Ermon, S. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 2022.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 1953.

Norris, J. R. Markov chains. *Cambridge university press*, 1998.

Peng, X., Guan, J., Liu, Q., and Ma, J. Moldiff: Addressing the atom-bond inconsistency problem in 3d molecule diffusion generation. *International Conference on Machine Learning*, 2023.

Pereira, J., Simpkin, A. J., Hartmann, M. D., Rigden, D. J., Keegan, R. M., and Lupas, A. N. High-accuracy protein structure prediction in casp14. *Proteins: Structure, Function, and Bioinformatics*, 89(12):1687–1699, 2021.

Qin, Y., Vignac, C., and Frossard, P. Sparse training of discrete diffusion models for graph generation. *arXiv preprint arXiv:2311.02142*, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *International conference on machine learning*, 2014.

Richemond, P. H., Dieleman, S., and Doucet, A. Categorical sdes with simplex diffusion. *arXiv preprint arXiv:2210.14784*, 2022.

Santos, J. E., Fox, Z. R., Lubbers, N., and Lin, Y. T. Blackout diffusion: Generative diffusion models in discrete-state spaces. *International Conference on Machine Learning*, 2023.

Shaul, N., Chen, R. T., Nickel, M., Le, M., and Lipman, Y. On kinetic optimal probability paths for generative models. *International Conference on Machine Learning*, 2023.

Shi, C., Wang, C., Lu, J., Zhong, B., and Tang, J. Protein sequence and structure co-design with equivariant translation. *arXiv preprint arXiv:2210.08761*, 2022.

Shih, A., Sadigh, D., and Ermon, S. Training and inference on any-order autoregressive models the right way. *Advances in Neural Information Processing Systems*, 2022.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning*, 2015.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2020.

Strudel, R., Tallec, C., Altché, F., Du, Y., Ganin, Y., Mensch, A., Grathwohl, W., Savinov, N., Dieleman, S., Sifre, L., et al. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022.

Sun, H., Dai, H., Dai, B., Zhou, H., and Schuurmans, D. Discrete langevin samplers via wasserstein gradient flow. *International Conference on Artificial Intelligence and Statistics*, 2023a.

Sun, H., Yu, L., Dai, B., Schuurmans, D., and Dai, H. Score-based continuous-time discrete diffusion models. *International Conference on Learning Representations*, 2023b.

Tang, Z., Gu, S., Bao, J., Chen, D., and Wen, F. Improved vector quantized diffusion models. *arXiv preprint arXiv:2205.16007*, 2022.

Trippe, B. L., Yim, J., Tischer, D., Baker, D., Broderick, T., Barzilay, R., and Jaakkola, T. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022.

van Kempen, M., Kim, S., Tumescheit, C., Mirdita, M., Söding, J., and Steinegger, M. Foldseek: fast and accurate protein structure search. *bioRxiv*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 2017.

Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. Digress: Discrete denoising diffusion for graph generation. *International Conference on Learning Representations*, 2023a.

Vignac, C., Osman, N., Toni, L., and Frossard, P. Midi: Mixed graph and 3d denoising diffusion for molecule generation. *arXiv preprint arXiv:2302.09048*, 2023b.

Vinothkumar, K. R. and Henderson, R. Structures of membrane proteins. *Quarterly reviews of biophysics*, 43(1): 65–158, 2010.

Wang, B. and Komatsuzaki, A. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, 2021.

Wang, H., Fu, T., Du, Y., Gao, W., Huang, K., Liu, Z., Chandak, P., Liu, S., Van Katwyk, P., Deac, A., et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

Watson, J. L., Juergens, D., Bennett, N. R., Trippe, B. L., Yim, J., Eisenach, H. E., Ahern, W., Borst, A. J., Ragotte, R. J., Milles, L. F., et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 2023.

Xu, Y., Deng, M., Cheng, X., Tian, Y., Liu, Z., and Jaakkola, T. Restart sampling for improving generative processes. *Advance in Neural Information Processing Systems*, 2023.

Yang, J. J., Yim, J., Barzilay, R., and Jaakkola, T. Fast non-autoregressive inverse folding with discrete diffusion. *arXiv preprint arXiv:2312.02447*, 2023.

Yi, K., Zhou, B., Shen, Y., Liò, P., and Wang, Y. G. Graph denoising diffusion for inverse protein folding. *arXiv preprint arXiv:2306.16819*, 2023.

Yim, J., Campbell, A., Foong, A. Y., Gastegger, M., Jiménez-Luna, J., Lewis, S., Satorras, V. G., Veeling, B. S., Barzilay, R., Jaakkola, T., et al. Fast protein backbone generation with se (3) flow matching. *arXiv preprint arXiv:2310.05297*, 2023a.

Yim, J., Trippe, B. L., De Bortoli, V., Mathieu, E., Doucet, A., Barzilay, R., and Jaakkola, T. Se (3) diffusion model with application to protein backbone generation. *arXiv preprint arXiv:2302.02277*, 2023b.

Zhang, P., Yin, H., Li, C., and Xie, X. Formulating discrete probability flow through optimal transport. *arXiv preprint arXiv:2311.03886*, 2023.

Zhang, Y. and Skolnick, J. Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic acids research*, 33(7):2302–2309, 2005.

Appendix to: Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design

A. Organization of Appendix

The Appendix is organized as follows. App. B provides proofs for all propositions in the main text. App. C analyses the cross entropy objective used to train DFM and links controlling the cross entropy to controlling the model log-likelihood. App. D discusses further related work. App. E shows how DFM can be applied to multidimensional data through applying factorization assumptions to $p_{t|1}$. App. F gives concrete realizations with PyTorch code for DFM using the masking or uniform forms for $p_{t|1}$. App. G discusses methods for sampling from CTMCs and discusses their relation to our sampling method. App. H compares DFM to classical discrete diffusion models in discrete and continuous time finding that they can be fit within the DFM framework. App. I gives further details and results for our text experiment. App. J gives further details and results for our protein co-design experiments.

B. Proofs

Notation When writing rate matrices, $R_t(i, j)$, we will assume $i \neq j$ unless otherwise explicitly stated.

We write $R_t(i) := \sum_{j \neq i} R_t(i, j)$.

B.1. Proof of Proposition 3.1

We simply take the expectation with respect to p_{data} of both sides of the Kolmogorov equation for $p_{t|1}(x_t|x_1)$ and $R_t(x_t, j|x_1)$. Note we use the fact that $R_t(i, i) = -\sum_{j \neq i} R_t(i, j)$ for compactness.

$$\begin{aligned}
\partial_t p_{t|1}(x_t|x_1) &= \sum_j R_t(j, x_t|x_1) p_{t|1}(j|x_1) \\
\mathbb{E}_{p_{\text{data}}(x_1)}\left[\partial_t p_{t|1}(x_t|x_1)\right] &= \mathbb{E}_{p_{\text{data}}(x_1)}\left[\sum_j R_t(j, x_t|x_1) p_{t|1}(j|x_1)\right] \\
\partial_t \mathbb{E}_{p_{\text{data}}(x_1)}\left[p_{t|1}(x_t|x_1)\right] &= \sum_j \sum_{x_1} p_{\text{data}}(x_1) p_{t|1}(j|x_1) R_t(j, x_t|x_1) \\
\partial_t p_t(x_t) &= \sum_j \sum_{x_1} p_t(j) p_{1|t}(x_1|j) R_t(j, x_t|x_1) \\
\partial_t p_t(x_t) &= \sum_j \mathbb{E}_{p_{1|t}(x_1|j)}\left[R_t(j, x_t|x_1)\right] p_t(j)
\end{aligned}$$

Where we notice that the final line is the Kolmogorov equation for a CTMC with marginals $p_t(x_t)$ and rate $\mathbb{E}_{p_{1|t}(x_1|x_t)}\left[R_t(x_t, j|x_t)\right]$. Therefore we have shown that $\mathbb{E}_{p_{1|t}(x_1|x_t)}\left[R_t(x_t, j|x_t)\right]$ generate $p_t(x_t)$.

B.2. Proof of Proposition 3.2

In the main text we provided the form for R_t^* under the assumption that $p_{t|1}(j|x_1) > 0$ for all j. Before proving Prop. 3.2, we first give the full form for R_t^*. First, assuming $x_t \neq j$ and $p_{t|1}(x_t|x_1) > 0$ we have,

$$R_t^*(x_t, j|x_1) := \frac{\text{ReLU}\left(\partial_t p_{t|1}(j|x_1) - \partial_t p_{t|1}(x_t|x_1)\right)}{\mathcal{Z}_t p_{t|1}(x_t|x_1)}$$

where $\text{ReLU}(a) = \max(a, 0)$ and $\mathcal{Z}_t$ is the number of states that have non-zero mass, $\mathcal{Z}_t = |\{x_t : p_{t|1}(x_t|x_1) > 0\}|$. $R_t^*(x_t, j|x_1) = 0$ when $p_{t|1}(x_t|x_1) = 0$ or $p_{t|1}(j|x_1) = 0$. When $x_t = j$, $R_t^*(x_t, x_t|x_1) = -\sum_{j \neq x_t} R_t^*(x_t, j|x_1)$ as we have defined before.

For our proof, we assume that $p_{t|1}(j|x_1) = 0 \implies \partial_t p_{t|1}(j|x_1) = 0$. This assumption means that when we have dead states with zero probability mass, they cannot be resurrected and gain probability mass in the future. We begin the proof with the Kolmogorov equation for processes conditioned on x_1,

$$\partial_t p_{t|1}(x_t|x_1) = \sum_{j \neq x_t} R_t(j, x_t|x_1) p_{t|1}(j|x_1) - \sum_{j \neq x_t} R_t(x_t, j|x_1) p_{t|1}(x_t|x_1) \tag{17}$$

We will now verify that R_t^* satisfies this Kolmogorov equation and thus generates the desired $p_{t|1}(x_t|x_1)$ conditional flow. We will first check that the Kolmogorov equation is satisfied when $p_{t|1}(x_t|x_1) > 0$. With this form of rate matrix, the RHS of equation (17) becomes

$$\begin{aligned}
\text{RHS} = &\sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \frac{\text{ReLU}\left(\partial_t p_{t|1}(x_t|x_1) - \partial_t p_{t|1}(j|x_1)\right)}{\mathcal{Z}_t p_{t|1}(j|x_1)} p_{t|1}(j|x_1) \\
&- \sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \frac{\text{ReLU}\left(\partial_t p_{t|1}(j|x_1) - \partial_t p_{t|1}(x_t|x_1)\right)}{\mathcal{Z}_t p_{t|1}(x_t|x_1)} p_{t|1}(x_t|x_1) \\
= &\frac{1}{\mathcal{Z}_t} \sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \text{ReLU}\left(\partial_t p_{t|1}(x_t|x_1) - \partial_t p_{t|1}(j|x_1)\right) \\
&- \frac{1}{\mathcal{Z}_t} \sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \text{ReLU}\left(\partial_t p_{t|1}(j|x_1) - \partial_t p_{t|1}(x_t|x_1)\right) \\
= &\frac{1}{\mathcal{Z}_t} \sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \left(\partial_t p_{t|1}(x_t|x_1) - \partial_t p_{t|1}(j|x_1)\right) \\
= &\frac{\mathcal{Z}_t - 1}{\mathcal{Z}_t} \partial_t p_{t|1}(x_t|x_1) - \frac{1}{\mathcal{Z}_t} \sum_{j \neq x_t, p_{t|1}(j|x_1)>0} \partial_t p_{t|1}(j|x_1) \\
= &\frac{\mathcal{Z}_t - 1}{\mathcal{Z}_t} \partial_t p_{t|1}(x_t|x_1) - \frac{1}{\mathcal{Z}_t} \partial_t (1 - p_{t|1}(x_t|x_1)) \\
= &\frac{\mathcal{Z}_t - 1}{\mathcal{Z}_t} \partial_t p_{t|1}(x_t|x_1) + \frac{1}{\mathcal{Z}_t} \partial_t p_{t|1}(x_t|x_1) \\
= &\partial_t p_{t|1}(x_t|x_1) \\
= &\text{LHS}
\end{aligned}$$

In the case that $p_{t|1}(x_t|x_1) = 0$ by assumption we have that $\partial_t p_{t|1}(x_t|x_1) = 0$. We have both $R_t^*(x_t, j|x_1) = 0$ and $R_t^*(j, x_t|x_1) = 0$ because $p_{t|1}(x_t|x_1) = 0$. Therefore we have LHS $=$ RHS $= 0$ and thus the Kolmogorov equation is satisfied.

Intuitively, we require the assumption that dead states cannot be resurrected because R_t^* is designed such that all states can equally distribute the mass flux requirements of making sure the marginal derivatives $\partial_t p_{t|1}(x_t|x_1)$ are satisfied. If there is a state for which $p_{t|1}(x_t|x_1) = 0$ but $\partial_t p_{t|1}(x_t|x_1) > 0$ then this state would require mass from other states but could not provide any mass of its own since $p_{t|1}(x_t|x_1) = 0$. This would then violate the sharing symmetry required for our form of R_t^*. We note that this assumption is not strictly satisfied for the masking interpolant at $t = 0$ or $t = 1$ and not satisfied for the uniform interpolant at $t = 1$. However, it is satisfied for any $t \in (0, 1)$ and so we can conceptualize starting our process at $t = \epsilon$, $\epsilon \ll 1$, $\epsilon > 0$, approximating a sample from $p_\epsilon(x_\epsilon)$ with a sample from $p_0(x_0)$ and running the process until $t = 1 - \epsilon$ and stopping here. The approximation can be made arbitrarily accurate by taking $\epsilon \to 0$.

B.3. Proof of Proposition 3.3

A rate matrix that satisfies the detailed balance condition (9) will result in $\partial_t p_{t|1}(i|x_1) = 0$ when simulating with this rate. This can be seen by substituting into the conditional Kolmogorov equation (17)

$$\partial_t p_{t|1}(x_t|x_1) = \sum_{j \neq x_t} R_t^{\text{DB}}(j, x_t|x_1) p_{t|1}(j|x_1)$$

$$
\begin{aligned}
&- \sum_{j \neq x_t} R_t^{\mathrm{DB}}(x_t, j|x_1) p_{t|1}(x_t|x_1) \\
\partial_t p_{t|1}(x_t|x_1) = &\sum_{j \neq x_t} R_t^{\mathrm{DB}}(x_t, j|x_1) p_{t|1}(x_t|x_1) \\
&- \sum_{j \neq x_t} R_t^{\mathrm{DB}}(x_t, j|x_1) p_{t|1}(x_t|x_1) \\
\partial_t p_{t|1}(x_t|x_1) = &0
\end{aligned}
$$

Given a rate matrix $R_t(x_t, j|x_1)$ that generates $p_{t|1}(x_t|x_1)$, we first prove that $R_t(x_t, j|x_1) + \eta R_t^{\mathrm{DB}}(x_t, j|x_1)$ also generates $p_{t|1}(x_t|x_1)$ for any $\eta \in \mathbb{R}^{\geq 0}$. We show this by verifying that the combined rate matrix satisfies the Kolmogorov equation for conditional flow $p_{t|1}(x_t|x_1)$. The right hand side of the Kolmogorov equation is

$$
\begin{aligned}
\text{RHS} &= \sum_j \left(R_t(x_t, j|x_1) + \eta R_t^{\mathrm{DB}}(x_t, j|x_1) \right) p_{t|1}(j|x_1) \\
&= \sum_j R_t(x_t, j|x_1) p_{t|1}(j|x_1) + \eta \underbrace{\sum_j R_t^{\mathrm{DB}}(x_t, j|x_1) p_{t|1}(j|x_1)}_{=0} \\
&= \sum_j R_t(x_t, j\ x_1) p_{t|1}(j|x_1) \\
&= \partial_t p_{t|1}(x_t|x_1) \\
&= \text{LHS}
\end{aligned}
$$

where we have used the fact that R^{DB} is in detailed balance with $p_{t|1}(j|x_1)$ and that $R_t(x_t, j|x_1)$ generates $p_{t|1}$. Since R_t^* is a matrix that generates $p_{t|1}$, we also have the stated result as a specific case: $R_t^* + \eta R_t^{\mathrm{DB}}$ generates $p_{t|1}$.

B.4. Proof of Proposition 3.4

We will assume we have D dimensional data $x_1^{1:D}$ with each $x_1^d \in \{1, \dots, S\}$. We give an overview of how our method operates in the multi-dimensional case in Appendix E. Namely, we assume that our conditional flow factorizes as $p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \prod_{d=1}^D p_{t|1}(x_t^d|x_1^d)$. We also assume that our rate matrix is 0 for jumps that vary more than 1 dimension at a time. Our optimality results are derived under these assumptions.

B.4.1. Masking Interpolant

We first prove that R_t^* achieves the minimum number of transitions for the masking interpolant case. We have

$$
p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \prod_{d=1}^D p_{t|1}(x_t^d|x_1^d)
$$

with

$$
p_{t|1}(x_t^d|x_1^d) = t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\delta\left\{x_t^d, M\right\}
$$

Our rate in dimension d is

$$
R_t^{*d}(x_t^d, j^d|x_1^d) = \begin{cases} \frac{\mathrm{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right)}{\mathcal{Z}_t^d p_{t|1}(x_t^d|x_1^d)} & \text{for } p_{t|1}(x_t^d|x_1^d) > 0, p_{t|1}(j^d|x_1^d) > 0 \\ = 0 & \text{otherwise} \end{cases}
$$

with $\mathcal{Z}_t^d = |\{j^d : p_{t|1}(j^d|x_1^d) > 0\}|$. Substituting in $\partial_t p_{t|1}$ and $p_{t|1}$ in the masking case gives

$$
R_t^{*d}(x_t^d, j^d|x_1^d) = \frac{1}{1-t}\delta\left\{x_t^d, M\right\}\delta\left\{j^d, x_1^d\right\}
$$

We refer to Appendix F.1 for the details of this derivation. Since R_t^{*d} depends only on x_t^d, j^d and x_1^d and not values in any other dimensions, each dimension propagates independently and we can consider each dimension in isolation. Consider the

process for dimension d. The CTMC begins in state $x_0^d = M$. We have $R_t^{*d}(x_t^d = M, j^d | x_1^d) = \frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}$. Therefore, the only possible next state that the process can jump to is x_1^d. Once the process has jumped to x_1^d, the rate then becomes $R_t^{*d}(x_t^d = x_1^d, j^d | x_1^d) = 0$. We also know that the process must jump because $p_1(x_t^d | x_1^d) = \delta\left\{x_t^d, x_1^d\right\}$, $x_1^d \neq M$ and we know our rate matrix traverses our desired marginals by Proposition 3.2. Therefore, exactly one jump is made in dimension d. In total, our D dimensional process will make D jumps. Under our factorization assumption, during a jump no more than one dimension can change value. Therefore, the absolute minimum number of jumps for any process that starts at $x_0^{1:D}$ with $x_0^d = M, \forall d$ and ends at $x_1^{1:D}$, $x_1^d \neq M, \forall d$ is D. Our prior distribution is $p_0(x_0^d) = \delta\left\{x_0^d, M\right\}$ and so for any x_0 sample, we will always need to make D jumps. Therefore, the minimum expected number of jumps is D and R_t^* achieves this minimum.

B.4.2. Uniform Interpolant

We now prove that R_t^* achieves the minimum number of transitions for the uniform interpolant case. The conditional flow is

$$p_{t|1}(x_t^d | x_1^d) = t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\frac{1}{S}$$

With this interpolant, our rate matrix becomes

$$R_t^{*d}(x_t^d, j^d | x_1^d) = \frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\left(1 - \delta\left\{x_t^d, x_1^d\right\}\right)$$

We refer to Appendix F.2 for the derivation. As before, R_t^{*d} depends only on the values in dimension d, x_t^d, j^d, x_1^d and therefore each process propagates independently in each dimension and we can consider each dimension in isolation. Considering dimension d, the process begins in state x_0^d. Both $x_0^d = x_1^d$ and $x_0^d \neq x_1^d$ are possible in the uniform interpolant case. In the case that $x_0^d = x_1^d$, then $R_t^{*d} = 0$ for all t and therefore no jumps are made in this dimension. In the case that $x_0^d \neq x_1^d$ then before any jump is made we have $R_t^{*d}(x_t^d, j^d | x_1^d) = \frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}$ and so the only possible next state the process can jump to is x_1^d. Once the process has jumped to x_1^d, the rate then becomes $R_t^*(x_t^d = x_1^d, j^d | x_1^d) = 0$ and so no more jumps are made. We also know that the process must jump at some point because $p_1(x_t^d | x_1^d) = \delta\left\{x_t^d, x_1^d\right\}$ and we know our rate matrix traverses our desired marginals by Proposition 3.2. Therefore, in the case that $x_0^d \neq x_1^d$, exactly one jump is made for the process in dimension d. In total, the number of jumps made in all D dimensions is $d_H(x_0, x_1) = |\{d : x_0^d \neq x_1^d\}|$ which is the Hamming distance between x_0 and x_1. The expected number of jumps for our process with R_t^* is thus $\mathbb{E}_{p_0(x_0)p_{\text{data}}(x_1)}\left[d_H(x_0, x_1)\right]$.

Now consider an optimal process that makes the minimum number of jumps when starting from x_0 and meets our factorization assumptions. By this assumption, during a jump only one dimension can change in value. Clearly we have that the minimum number of jumps required to get from x_0 to x_1 is $d_H(x_0, x_1)$. Therefore, for this optimal process we also have that the minimum number of expected jumps is $\mathbb{E}_{p_0(x_0)p_{\text{data}}(x_1)}\left[d_H(x_0, x_1)\right]$. Therefore, R_t^* achieves the minimum expected number of jumps.

B.4.3. Discussion

We have proven x_1 conditioned optimality only for the two simple conditional flows featured in the main text and we note that this result in not generally true for any conditional flow. Intuitively this is because R_t^* treats the distribution of mass symmetrically between states, considering only the local differences in $\partial_t p_{t|1}$ between pairs of states. In general, the optimal rate would need to solve a global programming problem.

We also note that although we have masking and uniform optimality for $R_t^*(x_t, j | x_1)$ when conditioned on x_1, this is not necessarily the case when we consider the unconditional version $\mathbb{E}_{p_{1|t}(x_1|x_t)}\left[R_t^*(x_t, j | x_1)\right]$. There may exist rate matrices that achieve a lower number of average jumps and successfully pass through the unconditional marginals $p_t(x_t) = \mathbb{E}_{p_{\text{data}}(x_1)}\left[p_{t|1}(x_t | x_1)\right]$. This is analogous to continuous flow-based methods which can create optimal straight-line paths when conditioned on the end point x_1, but don't necessarily achieve the optimal transport when considering the unconditional vector field (Shaul et al., 2023).

B.5. Proof of Proposition 4.1

We have that the individual velocities and rate matrices independently generate their respective flows in each dimension. Specifically, for $x_t^d \in \mathbb{R}^3$, we have that it satisfies the following Fokker-Planck equation,

$$\partial_t p_{t|1}(x_t^d|x_1^d) = -\nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(x_t^d|x_1^d)\right).$$

where $\nabla^{(d)}\cdot$ is the divergence operator for elements in dimension d. Similarly for $r_t \in \mathrm{SO}(3)$,

$$\partial_t p_{t|1}(r_t^d|r_1^d) = -\nabla^{(d)} \cdot \left(v_r^d(r_t^d|r_1^d) p_{t|1}(r_t^d|r_1^d)\right),$$

where the divergence operator now acts on elements in $\mathrm{Tan}_{r_t^d}\mathrm{SO}(3)$ (Yim et al., 2023b). Finally, for a_t^d, we have the familiar Kolmogorov equation,

$$\partial_t p_{t|1}(a_t^d|a_1^d) = \sum_j R_t(j, a_t^d|a_1^d) p_{t|1}^d(j|a_1^d).$$

For the joint space $\mathbf{T} \in (\mathbb{R}^3 \times \mathrm{SO}(3) \times \{1, \ldots, 20, M\})^D$ and process defined by the updates in Eq. (14), we also have a joint continuity equation known as the Fokker-Planck-Kolmogorov equation (Bect, 2010),

$$\begin{aligned}\partial_t p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) = &- \nabla \cdot \left(v_x(x_t^{1:D}|x_1^{1:D}) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) - \nabla \cdot \left(v_r(r_t^{1:D}|r_1^{1:D}) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) + \\ &\sum_{j^{1:D}} R_t(j^{1:D}, a_t^{1:D}|a_1^{1:D}) p_{t|1}\big((x_t^{1:D}, r_t^{1:D}, j^{1:D})|\mathbf{T}_1\big). \end{aligned} \tag{18}$$

Our aim is to show that the following choices of $p_{t|1}$, v_x, v_r and R_t corresponding to independent processes within each modality and dimension are consistent under Eq. (18) i.e. these choices of v_x, v_r and R_t will actually generate $p_{t|1}$ when simulated using Eq. (14) with $\Delta t \to 0$. The choices are as follows:

$$\begin{aligned} p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) &= \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d) p_{t|1}(r_t^d|r_1^d) p_{t|1}(a_t^d|a_1^d) \\ v_x(x_t^{1:D}|x_1^{1:D}) &= \begin{bmatrix} v_x^1(x_t^1|x_1^1) \\ \vdots \\ v_x^D(x_t^D|x_1^D) \end{bmatrix} \\ v_r(r^{1:D}|r^{1:D}) &= \begin{bmatrix} v_r^1(r_t^1|r_1^1) \\ \vdots \\ v_r^D(r_t^D|r_1^D) \end{bmatrix} \\ R_t(j^{1:D}, a_t^{1:D}|a_1^{1:D}) &= \sum_{d=1}^{D} \delta\left\{j^{1:D\setminus d}, a_t^{1:D\setminus d}\right\} R_t^d(j^d, a_t^d|a_1^d) \end{aligned}$$

More discussion regarding the form of $R_t(j^{1:D}, a_t^{1:D}|a_1^{1:D})$ can be found in Appendix E. Under these choices, the LHS of Eq. (18) becomes

$$\begin{aligned} \text{LHS} &= \partial_t p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) \\ &= \sum_{d=1}^{D} \partial_t p_{t|1}(x_t^d|x_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(x_t^d|x_1^d)} + \partial_t p_{t|1}(r_t^d|r_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(r_t^d|r_1^d)} + \partial_t p_{t|1}(a_t^d|a_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \end{aligned}$$

by the product rule for differentiation. The RHS of Eq. (18) becomes

$$\begin{aligned} \text{RHS} = &\sum_{d=1}^{D} -\nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) - \nabla^{(d)} \cdot \left(v_r^d(r_t^d|r_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) \\ &+ \sum_{j^{1:D}} \sum_{d=1}^{D} \delta\left\{j^{1:D\setminus d}, a_t^{1:D\setminus d}\right\} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}\big((x_t^{1:D}, r_t^{1:D}, j^{1:D}|\mathbf{T}_1\big) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{d=1}^{D} -\nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(x_t^d|x_1^d)\right) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(x_t^d|x_1^d)} - \nabla^{(d)} \cdot \left(v_r^d(r_t^d|r_1^d) p_{t|1}(r_t^d|r_1^d)\right) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(r_t^d|r_1^d)} \\
&\quad + \sum_{d=1}^{D} \left[\left\{ \prod_{d'=1}^{D} p_{t|1}^{d'}(x_t^{d'}|x_1^{d'}) p_{t|1}^{d'}(r_t^{d'}|r_1^{d'}) \right\} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d) \right. \\
&\qquad \left. \sum_{j^{1:D\setminus d}} \delta\left\{j^{1:D\setminus d}, a_t^{1:D\setminus d}\right\} \left\{ \prod_{d'=1\setminus d}^{D} p_{t|1}(j^{d'}|a_1^{d'}) \right\} \right] \\
&= \sum_{d=1}^{D} \partial_t p_{t|1}(x_t^d|x_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(x_t^d|x_1^d)} + \partial_t p_{t|1}(r_t^d|r_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(r_t^d|r_1^d)} \\
&\quad + \sum_{d=1}^{D} \left\{ \prod_{d'=1}^{D} p_{t|1}^{d'}(x_t^{d'}|x_1^{d'}) p_{t|1}^{d'}(r_t^{d'}|r_1^{d'}) \right\} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d) \left\{ \prod_{d'=1\setminus d}^{D} p_{t|1}(a_t^{d'}|a_1^{d'}) \right\} \\
&= \sum_{d=1}^{D} \partial_t p_{t|1}(x_t^d|x_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(x_t^d|x_1^d)} + \partial_t p_{t|1}(r_t^d|r_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(r_t^d|r_1^d)} + \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d) \\
&= \sum_{d=1}^{D} \partial_t p_{t|1}(x_t^d|x_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(x_t^d|x_1^d)} + \partial_t p_{t|1}(r_t^d|r_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(r_t^d|r_1^d)} + \partial_t p_{t|1}(a_t^d|a_1^d) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \\
&= \text{LHS}
\end{aligned}
$$

Therefore we have shown that our choices of v_x, v_r and R_t will generate the desired $p_{t|1}$.

B.6. Proof of Proposition 4.2

Our proof will mirror that of Proposition 3.1 by taking the expectation with respect to $p_{\text{data}}(\mathbf{T}_1)$ of both sides of the Fokker-Planck-Kolmogorov equation (Eq. (18)).

$$
\begin{aligned}
\mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[\partial_t p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right] =& \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\Big[-\nabla \cdot \left(v_x(x_t^{1:D}|x_1^{1:D}) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) - \nabla \cdot \left(v_r(r_t^{1:D}|r_1^{1:D}) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) + \\
& \sum_{j^{1:D}} R_t(j^{1:D}, a_t^{1:D}) p_{t|1}\big((x_t^{1:D}, r_t^{1:D}, j^{1:D})|\mathbf{T}_1\big)\Big] \\
\partial_t p_t(\mathbf{T}_t) =& \sum_{d=1}^{D} \Big\{ \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[-\nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right)\right] + \\
& \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[-\nabla^{(d)} \cdot \left(v_r^d(r_t^d|r_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right)\right] + \\
& \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[\frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d)\right] \Big\}
\end{aligned} \tag{19}
$$

where on the second line on the left hand side we have used the fact that $p_t(\mathbf{T}_t) = \mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right]$. We shall first examine the v_x^d term on the right hand side in isolation.

$$
\begin{aligned}
\mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)}\left[-\nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right)\right] &= -\int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} p_{\text{data}}(\mathbf{T}_1) \nabla^{(d)} \cdot \left(v_x^d(x_t^d|x_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)\right) \mathrm{d}x_1^{1:D} \mathrm{d}r_1^{1:D} \\
&= -\nabla^{(d)} \cdot \left(\int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} p_{\text{data}}(\mathbf{T}_1) v_x^d(x_t^d|x_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) \mathrm{d}x_1^{1:D} \mathrm{d}r_1^{1:D} \right) \\
&= -\nabla^{(d)} \cdot \left(\int_{x_1^d} v_x^d(x_t^d|x_1^d) \int_{x_1^{1:D\setminus d}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} p_{\text{data}}(\mathbf{T}_1) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) \mathrm{d}x_1^{1:D\setminus d} \mathrm{d}r_1^{1:D} \mathrm{d}x_1^d \right)
\end{aligned}
$$

$$= -\nabla^{(d)} \cdot \left(\int_{x_1^d} v_x^d(x_t^d|x_1^d) p_t(\mathbf{T}_t) p_{1|t}(x_1^d|\mathbf{T}_t) \right)$$
$$= -\nabla^{(d)} \cdot \left(\mathbb{E}_{p_{1|t}(x_1^d|\mathbf{T}_t)} \left[v_x^d(x_t^d|x_1^d) \right] p_t(\mathbf{T}_t) \right)$$

The same argiments follow through for the v_r^d term giving

$$\mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)} \left[-\nabla^{(d)} \cdot \left(v_r^d(r_t^d|r_1^d) p_{t|1}(\mathbf{T}_t|\mathbf{T}_1) \right) \right] = -\nabla^{(d)} \cdot \left(\mathbb{E}_{p_{1|t}(r_1^d|\mathbf{T}_t)} \left[v_r^d(r_t^d|r_1^d) \right] p_t(\mathbf{T}_t) \right)$$

We finally analyse the R_t^d term in isolation. In the following, we will use $a_t^{1:D\backslash d} \odot j^d$ to refer to the D dimensional discrete variable with the values $a_t^{1:D\backslash d}$ in all dimensions except d and the value j^d in dimension d.

$$\begin{aligned}
&\mathbb{E}_{p_{\text{data}}(\mathbf{T}_1)} \left[\frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d) \right] \\
&\quad = \int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} p_{\text{data}}(\mathbf{T}_1) \frac{p_{t|1}(\mathbf{T}_t|\mathbf{T}_1)}{p_{t|1}(a_t^d|a_1^d)} \sum_{j^d} R_t^d(j^d, a_t^d|a_1^d) p_{t|1}(j^d|a_1^d) \mathrm{d}x_1^{1:D} \mathrm{d}r_1^{1:D} \\
&\quad = \int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} \sum_{j^d} p_{t,1} \left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d), \mathbf{T}_1 \right) R_t^d(j^d, a_t^d|a_1^d) \mathrm{d}x_1^{1:D} \mathrm{d}r_1^{1:D} \\
&\quad = \int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D}} \sum_{j^d} p_t \left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) p_{1|t} \left(a_1^d | (x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) \\
&\qquad\qquad p \left((x_1^{1:D}, r_1^{1:D}, a_1^{1:D\backslash d}) | a_1^d, (x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) R_t^d(j^d, a_t^d|a_1^d) \mathrm{d}x_1^{1:D} \mathrm{d}r_1^{1:D} \\
&\quad = \sum_{j^d} p_t \left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) \sum_{a_1^d} p_{1|t} \left(a_1^d | (x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) R_t^d(j^d, a_t^d|a_1^d) \\
&\qquad \underbrace{\int_{x_1^{1:D}} \int_{r_1^{1:D}} \sum_{a_1^{1:D\backslash d}} p \left((x_1^{1:D}, r_1^{1:D}, a_1^{1:D\backslash d}) | a_1^d, (x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) \mathrm{d}x_1^{1:D} r_1^{1:D}}_{=1} \\
&\quad = \sum_{j^d} \mathbb{E}_{p_{1|t}(a_1^d|(x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d))} \left[R_t^d(j^d, a_t^d|a_1^d) \right] p_t \left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right)
\end{aligned}$$

Now we substitute these simplified forms back into Eq. (19).

$$\begin{aligned}
\partial_t p_t(\mathbf{T}_t) = \sum_{d=1}^{D} \Bigg\{ &- \nabla^{(d)} \cdot \left(\mathbb{E}_{p_{1|t}(x_1^d|\mathbf{T}_t)} \left[v_x^d(x_t^d|x_1^d) \right] p_t(\mathbf{T}_t) \right) - \nabla^{(d)} \cdot \left(\mathbb{E}_{p_{1|t}(r_1^d|\mathbf{T}_t)} \left[v_r^d(r_t^d|r_1^d) \right] p_t(\mathbf{T}_t) \right) + \\
&\sum_{j^d} \mathbb{E}_{p_{1|t}(a_1^d|(x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d))} \left[R_t^d(j^d, a_t^d|a_1^d) \right] p_t \left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\backslash d} \odot j^d) \right) \Bigg\}
\end{aligned} \tag{20}$$

We will now show that Eq. (20) is the Fokker-Planck-Kolmogorov equation for $p_t(\mathbf{T}_t)$ with the following choices for the velocities and rate matrix.

$$v_x(\mathbf{T}_t) = \begin{bmatrix} \mathbb{E}_{p_{1|t}(x_1^1|\mathbf{T}_t)} \left[v_x^1(x_t^1|x_1^1) \right] \\ \vdots \\ \mathbb{E}_{p_{1|t}(x_1^D|\mathbf{T}_t)} \left[v_x^D(x_t^D|x_1^D) \right] \end{bmatrix}$$
$$v_r(\mathbf{T}_t) = \begin{bmatrix} \mathbb{E}_{p_{1|t}(r_1^1|\mathbf{T}_t)} \left[v_r^1(r_t^1|r_1^1) \right] \\ \vdots \\ \mathbb{E}_{p_{1|t}(r_1^D|\mathbf{T}_t)} \left[v_r^D(r_t^D|r_1^D) \right] \end{bmatrix}$$

$$R_t(\mathbf{T}_t, j^{1:D}) = \sum_{d=1}^{D} \delta\left\{a_t^{1:D\setminus d}, j^{1:D\setminus d}\right\} \mathbb{E}_{p_{1|t}(a_1^d|\mathbf{T}_t)}\left[R_t^d(a_t^d, j^d|a_1^d)\right] \tag{21}$$

We shall substitute the choices in Eq. (21) into the Fokker-Planck-Kolmogorov equation for $p_t(\mathbf{T}_t)$ and show that this equals Eq. (20).

$$\begin{aligned}
\partial_t p_t(\mathbf{T}_t) &= -\nabla\cdot(v_x(\mathbf{T}_t)p_t(\mathbf{T}_t)) - \nabla\cdot(v_r(\mathbf{T}_t)p_t(\mathbf{T}_t)) + \sum_{j^{1:D}} R_t\left((x_t^{1:D}, r_t^{1:D}, j^{1:D}), a_t^{1:D}\right) p_t\left((x_t^{1:D}, r_t^{1:D}, j^{1:D})\right) \\
&= \sum_{d=1}^{D}\Bigg\{ -\nabla^{(d)}\cdot\left(\mathbb{E}_{p_{1|t}(x_1^d|\mathbf{T}_t)}\left[v_x^d(x_t^d|x_1^d)\right] p_t(\mathbf{T}_t)\right) - \nabla^{(d)}\cdot\left(\mathbb{E}_{p_{1|t}(r_1^d|\mathbf{T}_t)}\left[v_r^d(r_t^d|r_1^d)\right] p_t(\mathbf{T}_t)\right)\Bigg\} + \\
&\quad \sum_{j^{1:D}}\sum_{d=1}^{D} \delta\left\{j^{1:D\setminus d}, a_t^{1:D\setminus d}\right\} \mathbb{E}_{p_{1|t}(a_1^d|(x_t^{1:D}, r_t^{1:D}, j^{1:D}))}\left[R_t^d(j^d, a_t^d|a_1^d)\right] p_t\left((x_t^{1:D}, r_t^{1:D}, j^{1:D})\right) \\
&= \sum_{d=1}^{D}\Bigg\{ -\nabla^{(d)}\cdot\left(\mathbb{E}_{p_{1|t}(x_1^d|\mathbf{T}_t)}\left[v_x^d(x_t^d|x_1^d)\right] p_t(\mathbf{T}_t)\right) - \nabla^{(d)}\cdot\left(\mathbb{E}_{p_{1|t}(r_1^d|\mathbf{T}_t)}\left[v_r^d(r_t^d|r_1^d)\right] p_t(\mathbf{T}_t)\right) + \\
&\quad \sum_{j^d} \mathbb{E}_{p_{1|t}(a_1^d|(x_t^{1:D}, r_t^{1:D}, a_t^{1:D\setminus d}\odot j^d))}\left[R_t^d(j^d, a_t^d|a_1^d)\right] p_t\left((x_t^{1:D}, r_t^{1:D}, a_t^{1:D\setminus d}\odot j^d)\right)\Bigg\}
\end{aligned}$$

which we see matches Eq. (20). Therefore, we have shown that the choices for the velocities and rate matrix in Eq. (21) create a process that generates $p_t(\mathbf{T}_t)$ as desired.

C. Analysis of Training Objective

In this section we analyse how our cross entropy objective $\mathcal{L}_{\text{ce}}$ relates to the log-likelihood of the data under the generative model and to the ELBO used to train classical discrete diffusion models.

Our proof is structured as follows. We first introduce path space measures for CTMCs in Section C.1 that we will require for the rest of the derivation. In Section C.1.1 we then derive the standard evidence lower bound, $\mathcal{L}_{\text{ELBO}}$ on the model log likelihood, $\mathbb{E}_{p_{\text{data}}(x_1)}\left[\log p_\theta(x_1)\right]$. We then decompose $\mathcal{L}_{\text{ELBO}}$ into the cross entropy, a rate regularizer and a KL term in Section C.2. Finally in Section C.2.1 we show that $\mathcal{L}_{\text{ELBO}}$ corresponds exactly to the weighted cross entropy loss for the masking interpolant case.

C.1. Introduction to CTMC path measures

Before beginning the proof, we introduce path space measures for CTMC processes, following the exposition in (Del Moral & Penev, 2017), Chapter 18. A path of a CTMC is a single trajectory from time 0 to time t. The trajectory is a function $\omega : s \in [0, t] \mapsto \omega_s \in \{1, \dots, S\}$ that is everywhere right continuous and has left limits everywhere (also known as càdlàg paths). Intuitively, it is a function that takes in a time variable and outputs the position of the particle following the trajectory at that time. The càdlàg condition in our case states that at jump time τ we have ω_τ taking the new jumped to value and $\omega_\tau^- := \lim_{s\uparrow\tau}\omega_s$ being the previous value before the jump, see Fig. 1B.

A trajectory drawn from the CTMC, W, can be fully described through its jump times, $T_1, \dots T_n$ and its state values between jumps, $W_0, W_1, \dots, W_{T_n}$ where at jump time T_k the CTMC jumps from state value W_{k-1} to value W_k. A path space measure $\mathbb{P}$ is able to assign probabilities to a drawn trajectory W from time 0 to t in the sense of

$$\mathbb{P}(W \in \mathrm{d}\omega) := \mathbb{P}\left(W_0 \in \mathrm{d}\omega_0, (T_1, W_{T_1}) \in \mathrm{d}(t_1, \omega_{t_1}), \dots (T_n, W_{T_n}) \in \mathrm{d}(t_n, \omega_{t_n}), T_{n+1} \geq t\right)$$

where $\mathrm{d}\omega_{t_n}$ and $\mathrm{d}t_n$ denote infinitesimal neighborhoods around the points $\omega_{t_n} \in \{1, \dots, S\}$ and $t_n \in [0, t]$. This is the same sense in which a probability density function assigns probabilities to the infinitesimal neighborhood around a continuous valued variable.

To understand the form of $\mathbb{P}(W \in \mathrm{d}\omega)$ we remind ourselves of the definition of a CTMC with rate matrix R_t. The CTMC waits in the current state for an amount of time determined by an exponential random variable with time-inhomogeneous

rate $R_t(W_t) := \sum_{k \neq W_t} R_t(W_t, k)$, see Norris (1998) and Campbell et al. (2022) Appendix A for more details. After the wait time is finished, the CTMC jumps to a next chosen state where the jump distribution is

$$\mathbb{P}(W_{t_k}|W_{t_k}^-) = \frac{R_t(W_{t_k}^-, W_{t_k})\left(1 - \delta\left\{W_{t_k}^-, W_{t_k}\right\}\right)}{R_t(W_{t_k}^-)}$$

For an exponential random variable with time-inhomogeneous rate, the cumulative distribution function is given by

$$\mathbb{P}(T < t) = 1 - \exp\left(-\int_{s=0}^{s=t} R_s(W_s^-)\mathrm{d}s\right)$$

Therefore, the probability density function, $p(t) = \frac{\partial}{\partial t}\mathbb{P}(T < t)$, is

$$p(t) = \exp\left(-\int_{s=0}^{s=t} R_s(W_s^-)\mathrm{d}s\right) R_t(W_t^-)$$

We finally note that if we wish to know $\mathbb{P}(T_k < t|T_{k-1})$ i.e. the probability that the k-th jump time is less than t given we know the $k-1$-th jump time, then this is just an exponential random variable started at time T_{k-1} when the previous jump occurred,

$$\mathbb{P}(T_k < t|T_{k-1}) = 1 - \exp\left(-\int_{s=T_{k-1}}^{s=t} R_s(W_s^-)\mathrm{d}s\right)$$

In other words, we simply start a new exponential timer once the previous jump occurs and the same equation carries through.

We can now write the form of $\mathbb{P}(W \in \mathrm{d}\omega)$. We split it into a series of conditional distributions

$$\begin{aligned}\mathbb{P}(W \in \mathrm{d}\omega) =& \mathbb{P}(W_0 \in \mathrm{d}\omega_0)\mathbb{P}((T_1, W_{T_1} \in \mathrm{d}(t_1, \omega_{t_1})|W_0) \times \dots \\ &\times \mathbb{P}((T_n, W_{T_n}) \in \mathrm{d}(t_n, \omega_{t_n})|W_0, (T_1, W_{T_1}), \dots, (T_{n-1}, W_{T_{n-1}})) \\ &\times \mathbb{P}(T_{n+1} \geq t|W_0, (T_1, W_{T_1}), \dots, (T_n, W_{T_n}))\end{aligned}$$

$$\begin{aligned}\mathbb{P}(W \in \mathrm{d}\omega) =& p_0(W_0)\exp\left(-\int_{s=0}^{s=T_1} R_s(W_s^-)\mathrm{d}s\right) R_{T_1}(W_{T_1}^-)\mathbb{P}(W_{T_1}|W_{T_1}^-) \times \dots \\ &\times \exp\left(-\int_{s=T_{n-1}}^{s=T_n} R_s(W_s^-)\mathrm{d}s\right) R_{T_n}(W_{T_n}^-)\mathbb{P}(W_{T_n}|W_{T_n}^-)\exp\left(-\int_{s=T_n}^{s=t} R_s(W_s^-)\mathrm{d}s\right) \\ \mathbb{P}(W \in \mathrm{d}\omega) =& p_0(W_0)\exp\left(-\int_{s=0}^{s=t} R_s(W_s^-)\mathrm{d}s\right) \prod_{s:W_s \neq W_s^-} R_s(W_s^-, W_s)\end{aligned}$$

where p_0 is the initial state distribution.

We will also need to understand Girsanov's transformation for CTMCs. Girsanov's transformation can be thought of as 'importance sampling' for path space measures. Specifically, if we take an expectation with respect to path measure $\mathbb{P}$, $\mathbb{E}_{\mathbb{P}}[f(W)]$, then this is equal to $\mathbb{E}_{\mathbb{Q}}\left[f(W)\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}(W)\right]$ where $\mathbb{Q}$ is a different path measure and $\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}$ is known as the Radon-Nikodym derivative. The path measure $\mathbb{Q}$ will result from considering a CTMC with a different rate matrix to our original measure $\mathbb{P}$. Girsanov's transformation allows us to calculate the expectation which should have been taken with respect to the CTMC with $\mathbb{P}$ rate matrix instead with a CTMC with rate matrix corresponding to $\mathbb{Q}$.

The Radon-Nikodym derivative in our case has a form that is simply the ratio of $\mathbb{P}(W \in \mathrm{d}\omega)$ and $\mathbb{Q}(W \in \mathrm{d}\omega)$. Let R_t, p_0 be the rate matrix and initial distribution defining $\mathbb{P}$ and let R'_t, p'_0 be the rate matrix and initial distribution defining $\mathbb{Q}$.

$$\frac{\mathrm{d}\mathbb{P}}{\mathrm{d}\mathbb{Q}}(W) = \frac{p_0(W_0)\exp\left(-\int_{s=0}^{s=t} R_s(W_s^-)\mathrm{d}s\right)\prod_{s:W_s \neq W_s^-} R_s(W_s^-, W_s)}{p'_0(W_0)\exp\left(-\int_{s=0}^{s=t} R'_s(W_s^-)\mathrm{d}s\right)\prod_{s:W_s \neq W_s^-} R'_s(W_s^-, W_s)}$$

C.1.1. Derivation of $\mathcal{L}_{\text{ELBO}}$

In this section we will derive the standard evidence lower bound for the model log-likelihood assigned to the data, $\mathbb{E}_{p_{\text{data}}(x_1)}[\log p_\theta(x_1)]$ when using our learned generative process to generate data. The entire structure of this section can be understood intuitively by making analogy to the derivation of the evidence lower bound for VAEs, (Kingma & Welling, 2013; Rezende et al., 2014; Huang et al., 2021). In a VAE, we have a latent variable model $p_\theta(z, x)$ for observed data x. To derive the ELBO, we introduce a second distribution over the latent variables $q(z|x)$ with which we will use to take the expectation. The ELBO derivation proceeds as

$$\log p_\theta(x) = \log \sum_z p_\theta(z, x)$$

$$\log p_\theta(x) = \log \sum_z q(z|x) \frac{p_\theta(z, x)}{q(z|x)} \quad \text{Girsanov's transformation / Importance sampling}$$

$$\log p_\theta(x) \geq \sum_z q(z|x) \log \left(\frac{p_\theta(z, x)}{q(z|x)} \right) \quad \text{Jensen's inequality}$$

$$\mathbb{E}_{p_{\text{data}}(x)}[p_\theta(x)] \geq \mathbb{E}_{p_{\text{data}}(x)q(z|x)}[\log p_\theta(z, x)] + C$$

In our case, x corresponds to the final state of the generative process at time $t = 1$, x_1. The latent variable z corresponds to all other states of the CTMC W_t, $t \in [0, 1)$. Our model $p_\theta(z, x)$ corresponds to our generative CTMC with rate matrix $R_t^\theta(x_t, j) = \mathbb{E}_{p_\theta(x_1|x_t)}[R_t(x_t, j|x_1)]$ and initial distribution $p_0(x_0)$. Our latent variable distribution $q(z|x)$ corresponds to the x_1 conditioned CTMC that begins at distribution $p_{0|1}(x_0|x_1)$ and simulates with x_1 conditioned rate matrix $R_t(x_t, j|x_1)$. We note here that $R_t(x_t, j|x_1)$ can be any rate matrix that generates the desired x_1 conditional flow, $p_{t|1}(x_t|x_1)$ as we described in the main text.

We now derive $\mathcal{L}_{\text{ELBO}}$ using our path space measures for CTMCs. We will use $\mathbb{P}^\theta$ to denote the path measure corresponding to the CTMC simulating from $p_0(x_0)$ using the generative rate matrix $R_t^\theta(x_t, j) = \mathbb{E}_{p_\theta(x_1|x_t)}[R_t(x_t, j|x_1)]$. We will use $\mathbb{Q}^{|x_1}$ to denote the path measure corresponding to the CTMC simulating from $p_{0|1}(x_0|x_1)$ using the x_1 conditioned rate matrix $R_t(x_t, j|x_1)$.

We begin by marginalizing out the latent variables, W_t, $t \in [0, 1)$ for our generative CTMC

$$\log p_\theta(x_1) = \log \int_{W_1 = x_1} \mathbb{P}^\theta(\mathrm{d}\omega)$$

We now apply Girsnov's transformation using our x_1 conditioned CTMC

$$\log p_\theta(x_1) = \log \int_{W_1 = x_1} \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \frac{\mathrm{d}\mathbb{P}^\theta}{\mathrm{d}\mathbb{Q}^{|x_1}}(\omega)$$

where

$$\frac{\mathrm{d}\mathbb{P}^\theta}{\mathrm{d}\mathbb{Q}^{|x_1}}(\omega) = \frac{p_0(W_0) \exp\left(- \int_{t=0}^{t=1} R_t^\theta(W_t^-)\mathrm{d}t \right) \prod_{t:W_t \neq W_t^-} R_t^\theta(W_t^-, W_t)}{p_{0|1}(W_0|x_1) \exp\left(- \int_{t=0}^{t=1} R_t(W_t^-|x_1)\mathrm{d}t \right) \prod_{t:W_t \neq W_t^-} R_t(W_t^-, W_t|x_1)}$$

we note at this point that $p_{0|1}(W_0|x_1) = p_0(W_0)$ and the two intial distribution terms cancel out. Now, apply Jensen's inequality

$$\log p_\theta(x_1) \geq \int_{W_1 = x_1} \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \log \frac{\mathrm{d}\mathbb{P}^\theta}{\mathrm{d}\mathbb{Q}^{|x_1}}(\omega)$$

and take the expectation with respect to the data distribution

$$\mathbb{E}_{p_{\text{data}}(x_1)}[\log p_\theta(x_1)] \geq \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \log \frac{\mathrm{d}\mathbb{P}^\theta}{\mathrm{d}\mathbb{Q}^{|x_1}}(\omega)$$

Finally, substitute in the form for $\frac{\mathrm{d}\mathbb{P}^\theta}{\mathbb{Q}^{|x_1}}$ and take terms that don't depend on θ out into a constant

$$\mathbb{E}_{p_{\text{data}}(x_1)}[\log p_\theta(x_1)] \geq \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \left\{ - \int_{t=0}^{t=1} R_t^\theta(W_t^-)\mathrm{d}t + \sum_{t:W_t \neq W_t^-} \log R_t^\theta(W_t^-, W_t) \right\} + C$$

$$= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\left\{-\int_{t=0}^{t=1} R_t^\theta(W_t^-)\mathrm{d}t + \sum_{t:W_t\neq W_t^-} \log \mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right\} + C$$

$$= \mathcal{L}_{\text{ELBO}} + C$$

where

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\left\{-\int_{t=0}^{t=1} R_t^\theta(W_t^-)\mathrm{d}t + \sum_{t:W_t\neq W_t^-} \log \mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right\} \tag{22}$$

C.2. Decomposition of $\mathcal{L}_{\text{ELBO}}$

Consider the term $\log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right)$,

$$\begin{aligned}
\log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right) &= \log\left(\mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right]\right) \\
&= \log\left(\mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right]\right) \\
&\quad + \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right)\right] \\
&\quad - \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right)\right] \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] + C \\
&\quad + \log\left(\mathbb{E}_{p(\tilde{x}|W_t^-)}\left[\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right]\right) \\
&\quad - \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-, W_t|\tilde{x}_1)\right)\right] \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] + C \\
&\quad + \log\left(\mathbb{E}_{p(\tilde{x}|W_t^-)}\left[\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}\mathbb{P}(W_t|W_t^-, \tilde{x}_1)R_t(W_t^-|\tilde{x}_1)\right]\right) \\
&\quad - \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(\tilde{x}_1|W_t^-)}{p(\tilde{x}_1|W_t^-)}\mathbb{P}(W_t|W_t^-, \tilde{x}_1)R_t(W_t^-|\tilde{x}_1)\right)\right]
\end{aligned}$$

where we have used our definition of the jump distribution of

$$\mathbb{P}(W_t|W_t^-, \tilde{x}_1) = \frac{R_t(W_t^-, W_t|\tilde{x}_1)}{R_t(W_t^-|\tilde{x}_1)}$$

Now we define two new distributions,

$$p_\theta(\tilde{x}_1|W_t^-)\mathbb{P}(W_t|W_t^-, \tilde{x}_1) = p_\theta(W_t|W_t^-)p_\theta(\tilde{x}_1|W_t^-, W_t)$$

where

$$p_\theta(W_t|W_t^-) := \sum_{\tilde{x}_1} p_\theta(\tilde{x}_1|W_t^-)\mathbb{P}(W_t|W_t^-, \tilde{x}_1)$$

and

$$p_\theta(\tilde{x}_1|W_t^-, W_t) := \frac{p_\theta(\tilde{x}_1|W_t^-)\mathbb{P}(W_t|W_t^-, \tilde{x}_1)}{\sum_{x_1'} p_\theta(x_1'|W_t^-)\mathbb{P}(W_t|W_t^-, x_1')} \tag{23}$$

Substitute in these newly defined distributions into our equation for $\log\left(\mathbb{E}_{p_\theta(\tilde{x}|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right)$ to get

$$\begin{aligned}
\log\left(\mathbb{E}_{p_\theta(\tilde{x}|W_t^-)}\left[R_t(W_t^-, W_t|\tilde{x}_1)\right]\right) &= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] + C \\
&\quad + \log\left(\mathbb{E}_{p(\tilde{x}|W_t^-)}\left[\frac{p_\theta(W_t|W_t^-)p_\theta(\tilde{x}_1|W_t^-, W_t)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-|\tilde{x}_1)\right]\right) \\
&\quad - \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(W_t|W_t^-)p_\theta(\tilde{x}_1|W_t^-, W_t)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-|\tilde{x}_1)\right)\right] \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] + C \\
&\quad + \cancel{\log p_\theta(W_t|W_t^-)} + \log\left(\mathbb{E}_{p(\tilde{x}|W_t^-)}\left[\frac{p_\theta(\tilde{x}_1|W_t^-, W_t)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-|\tilde{x}_1)\right]\right) \\
&\quad - \cancel{\log p_\theta(W_t|W_t^-)} - \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\left(\frac{p_\theta(\tilde{x}_1|W_t^-, W_t)}{p(\tilde{x}_1|W_t^-)}R_t(W_t^-|\tilde{x}_1)\right)\right] \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] \\
&\quad + \log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-, W_t)}\left[R_t(W_t^-|\tilde{x}_1)\right]\right) \\
&\quad + \mathrm{KL}\left(p(\tilde{x}_1|W_t^-)\,||\,p_\theta(\tilde{x}_1|W_t^-, W_t)\right) + C
\end{aligned}$$

Substituting this into our form for $\mathcal{L}_{\text{ELBO}}$ given in equation (22) gives

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\Bigg\{ - \int_{t=0}^{t=1} R_t^\theta(W_t^-)\mathrm{d}t + \sum_{t:W_t \neq W_t^-}\Bigg(&\mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] + \\
&\log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-, W_t)}\left[R_t(W_t^-|\tilde{x}_1)\right]\right) + \\
&\mathrm{KL}\left(p(\tilde{x}_1|W_t^-)\,||\,p_\theta(\tilde{x}_1|W_t^-, W_t)\right)\Bigg)\Bigg\}
\end{aligned}$$

Substituting this into our original bound on the model log-likelihood gives

$$\mathbb{E}_{p_{\text{data}}(x_1)}\left[\log p_\theta(x_1)\right] \geq \mathcal{L}_{\text{ELBO}} + C = \mathcal{L}_{\text{ce}} + \mathcal{L}_R + \mathcal{L}_{\text{KL}} + C$$

where

$$\begin{aligned}
\mathcal{L}_{\text{ce}} &= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \sum_{t:W_t^- \neq W_t} \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] \\
\mathcal{L}_R &= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\Bigg\{ - \int_{t=0}^{t=1} R_t^\theta(W_t)\mathrm{d}t + \sum_{t:W_t^- \neq W_t} \log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-, W_t)}\left[R_t(W_t^-|\tilde{x}_1)\right]\right)\Bigg\} \\
\mathcal{L}_{\text{KL}} &= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \sum_{t:W_t^- \neq W_t} \mathrm{KL}\left(p(\tilde{x}_1|W_t^-)\,||\,p_\theta(\tilde{x}_1|W_t^-, W_t)\right)
\end{aligned}$$

and C is a constant term independent of θ.

In the next stages of the proof, we going to show that $\mathcal{L}_{\text{ce}}$ is the weighted cross-entropy, $\mathcal{L}_R$ is a regularizer towards the arbitrarily chosen x_1 conditioned rate matrix that we argue we can ignore and $\mathcal{L}_{\text{KL}}$ is a KL term that we will absorb into the bound on the model log-likelihood.

In order to proceed, we will need to make use of Dynkin's formula

$$\int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \sum_{t:W_t^- \neq W_t} f(W_t^-, W_t) = \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \int_{t=0}^{t=1} \sum_{y \neq W_t} R_t(W_t, y|x_1) f(W_t, y)\mathrm{d}t$$

where $f(\cdot,\cdot)$ is a two-argument function. This formula can be understood intuitively as allowing us to switch from a sum over the jump times to a full integral over the time interval appropriately weighted by the probability that a jump occurs and the destination to which a jump goes to.

Weighted Cross Entropy We first show that $\mathcal{L}_{\text{ce}}$ is the weighted cross entropy.

$$
\begin{aligned}
\mathcal{L}_{\text{ce}} &= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \sum_{t:W_t^- \neq W_t} \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right] \\
&= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega) \int_{t=0}^{t=1} \sum_{y \neq W_t} R_t(W_t, y|x_1)\mathbb{E}_{p(\tilde{x}_1|W_t)}\left[\log p_\theta(\tilde{x}_1|W_t)\right]\mathrm{d}t \qquad \text{Dynkin} \\
&= \int\int_{t=0}^{t=1} p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\mathbb{E}_{p(\tilde{x}_1|W_t)}\left[\log p_\theta(\tilde{x}_1|W_t)\right] R_t(W_t|x_1)\mathrm{d}t \\
&= \mathbb{E}_{p_{\text{data}}(x_1)\mathcal{U}(t;0,1)p(x_t|x_1)}\left[R_t(x_t|x_1)\mathbb{E}_{p(\tilde{x}_1|x_t)}\left[\log p_\theta(\tilde{x}_1|x_t)\right]\right] \\
&= \mathbb{E}_{p_{\text{data}}(x_1)\mathcal{U}(t;0,1)p(x_t|x_1)p(\tilde{x}_1|x_t)}\left[R_t(x_t|x_1)\log p_\theta(\tilde{x}_1|x_t)\right] \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p(x_1,x_t)p(\tilde{x}_1|x_t)}\left[R_t(x_t|x_1)\log p_\theta(\tilde{x}_1|x_t)\right] \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p(x_t)p(x_1|x_t)p(\tilde{x}_1|x_t)}\left[R_t(x_t|x_1)\log p_\theta(\tilde{x}_1|x_t)\right] \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p(x_t)p(\tilde{x}_1|x_t)p(x_1|x_t)}\left[R_t(x_t|\tilde{x}_1)\log p_\theta(x_1|x_t)\right] \qquad \text{Relabel } x_1 \leftrightarrow \tilde{x}_1 \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p(x_t)p(x_1|x_t)}\left[\mathbb{E}_{p(\tilde{x}_1|x_t)}\left[R_t(x_t|\tilde{x}_1)\right]\log p_\theta(x_1|x_t)\right] \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p(x_t)p(x_1|x_t)}\left[\omega_t(x_t)\log p_\theta(x_1|x_t)\right]
\end{aligned}
$$

where on the second line we apply Dynkin's formula with $f(W_t^-, W_t) = \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log p_\theta(\tilde{x}_1|W_t^-)\right]$ which we note is independent of W_t. $\omega_t(x_t)$ is a weighting function. In diffusion model training it is common for the likelihood based objective to be a weighted form of a recognisable loss e.g. the L2 loss for diffusion models. Here we have a 'likelihood weighted' cross entropy. We can then make the same approximation as in diffusion models and set $\omega(x_t) = 1$ to equally weight all loss levels. This also has the benefit of making our loss independent of the arbitrarily chosen rate matrix R_t that could have been any rate that generates the desired conditional flow.

Rate Forcing Term We now analyse the term $\mathcal{L}_R$. We will show that it is approximately equal to an objective which at its optimum sets the learned generative rate matrix to have the same overall jump probability as the arbitrarily chosen rate matrix that generates our $p_{t|1}(x_t|x_1)$ conditional flow.

$$
\begin{aligned}
\mathcal{L}_R &= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\left\{ -\int_{t=0}^{t=1} R_t^\theta(W_t)\mathrm{d}t + \sum_{t:W_t^- \neq W_t} \log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-, W_t)}\left[R_t(W_t^-|\tilde{x}_1)\right]\right)\right\} \\
&= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\left\{ -\int_{t=0}^{t=1} R_t^\theta(W_t)\mathrm{d}t + \int_{t=0}^{t=1}\sum_{y \neq W_t} R_t(W_t, y|x_1)\log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t, y)}\left[R_t(W_t|\tilde{x}_1)\right]\right)\mathrm{d}t\right\}
\end{aligned}
$$

where on the second line we have applied Dynkin's formula with $f(W_t^-, W_t) = \mathbb{E}_{p_\theta(\tilde{x}_1|W_t^-, W_t)}\left[R_t(W_t^-|\tilde{x}_1)\right]$. To further understand this term, we make the following approximation

$$
\mathbb{E}_{p_\theta(\tilde{x}_1|W_t, y)}\left[R_t(W_t|\tilde{x}_1)\right] \approx \mathbb{E}_{p_\theta(\tilde{x}_1|W_t)}\left[R_t(W_t|\tilde{x}_1)\right]
$$

$p_\theta(\tilde{x}_1|W_t, y)$ is the Bayesian posterior update given by equation (23) starting with prior $p_\theta(\tilde{x}_1|W_t)$ and with likelihood $\mathbb{P}(y|W_t, \tilde{x}_1)$. It is therefore the models prediction of $\tilde{x}_1$ updated with the information that the process has jumped to new value y. When our CTMC is multi-dimensional then a single jump will change only a single dimension, see Appendix E, and so when we operate in high-dimensional settings, the Bayesian posterior will be close to the prior.

We will denote the approximate form of $\mathcal{L}_R$ as $\hat{\mathcal{L}}_R$.

$$
\hat{\mathcal{L}}_R = \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\left\{ -\int_{t=0}^{t=1} R_t^\theta(W_t)\mathrm{d}t + \int_{t=0}^{t=1}\sum_{y \neq W_t} R_t(W_t, y|x_1)\log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t)}\left[R_t(W_t|\tilde{x}_1)\right]\right)\mathrm{d}t\right\}
$$

$$
\begin{aligned}
&= \int p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\Bigg\{ -\int_{t=0}^{t=1} R_t^\theta(W_t)\mathrm{d}t + \int_{t=0}^{t=1} \log\left(\mathbb{E}_{p_\theta(\tilde{x}_1|W_t)}\left[R_t(W_t|\tilde{x}_1)\right]\right) R_t(W_t|x_1)\mathrm{d}t \Bigg\} \\
&= \int\int_{t=0}^{t=1} p_{\text{data}}(\mathrm{d}x_1)\mathbb{Q}^{|x_1}(\mathrm{d}\omega)\Bigg\{ -R_t^\theta(W_t) + R_t(W_t|x_1)\log R_t^\theta(W_t)\Bigg\}\mathrm{d}t \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p_{\text{data}}(x_1)p_t(x_t|x_1)}\left[-R_t^\theta(x_t) + R_t(x_t|x_1)\log R_t^\theta(x_t)\right] \\
&= \mathbb{E}_{\mathcal{U}(t;0,1)p_t(x_t)}\left[-R_t^\theta(x_t) + \mathbb{E}_{p(x_1|x_t)}\left[R_t(x_t|x_1)\right]\log R_t^\theta(x_t)\right]
\end{aligned}
$$

where on the third line we have used the definition of $R_t^\theta(W_t) = \mathbb{E}_{p_\theta(\tilde{x}_1|W_t)}\left[R_t(W_t|\tilde{x}_1)\right]$. Now consider maximizing $\hat{\mathcal{L}}_R$ with respect to the value of $R_\tau^\theta(z)$ at test input z and test time τ. Differentiating $\hat{\mathcal{L}}_R$ with respect to $R_\tau^\theta(z)$ and setting to 0 gives

$$
\begin{aligned}
\frac{\partial\hat{\mathcal{L}}_R}{\partial R_\tau(z)} &= p_\tau(z)\left(-1 + \mathbb{E}_{p(x_1|z)}\left[R_\tau(z|x_1)\right]\frac{1}{R_\tau^\theta(z)}\right) = 0 \\
&\implies R_\tau^\theta(z) = \mathbb{E}_{p(x_1|z)}\left[R_\tau(z|x_1)\right] \quad \text{at stationarity}
\end{aligned}
$$

Therefore, we have found that maximizing $\hat{\mathcal{L}}_R$ encourages $R_t^\theta(x_t)$ to equal $\mathbb{E}_{p(x_1|x_t)}\left[R_t(x_t|x_1)\right]$. However, $R_t(x_t|x_1)$ is the overall rate of jumps for the arbitrarily chosen rate matrix that generates the $p_{t|1}(x_t|x_1)$ conditional flow. This rate of jumps is completely dependent on the level of stochasticity chosen for $R_t(x_t|x_1)$ which does not have any a priori known correct level. Therefore, we do not want to be encouraging our learned generative rate matrix R_t^θ to be matching this stochasticity level and so the term $\hat{\mathcal{L}}_R$ is undesirable to have in the objective. The true evidence lower bound includes the term $\mathcal{L}_R$ which we expect to have a similar effect as $\hat{\mathcal{L}}_R$ as we argued previously.

KL Term When we maximize the $\mathcal{L}_{\text{ELBO}}$ objective, we would try to maximize the $\mathcal{L}_{\text{KL}}$ term i.e. we try and push $p(\tilde{x}_1|W_t^-)$ and $p_\theta(\tilde{x}_1|W_t^-, W_t)$ as far apart as possible. This makes sense to do as we try and push the posterior over $\tilde{x}_1$ given the information contained in both the pre-jump state W_t^- and the post jump state W_t away from the distribution over $\tilde{x}_1$ given just the information within W_t^-. Digging into this term deeper we see that

$$
\begin{aligned}
&\text{KL}\left(p(\tilde{x}_1|W_t^-)\,||\,p_\theta(\tilde{x}_1|W_t^-, W_t)\right) \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[\log\frac{p(\tilde{x}_1|W_t^-)}{p_\theta(\tilde{x}_1|W_t^-, W_t)}\right] \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[-\log\left(p_\theta(\tilde{x}_1|W_t^-)\mathbb{P}(W_t|W_t^-, \tilde{x}_1)\right) + \log\left(\sum_{x_1'} p_\theta(x_1'|W_t^-)\mathbb{P}(W_t|W_t^-, x_1')\right)\right] + C \\
&= \mathbb{E}_{p(\tilde{x}_1|W_t^-)}\left[-\log p_\theta(\tilde{x}_1|W_t^-) + \log\left(\sum_{x_1'} p_\theta(x_1'|W_t^-)\mathbb{P}(W_t|W_t^-, x_1')\right)\right] + C
\end{aligned}
$$

where we have substituted in our definition of $p_\theta(\tilde{x}_1|W_t^-, W_t)$ given by equation (23). We see that the first term $-\log p_\theta(\tilde{x}_1|W_t^-)$ cancels with our cross entropy term. This then makes clear how we have arrived at our cross entropy decomposition of $\mathcal{L}_{\text{ELBO}}$. $\mathcal{L}_{\text{ELBO}}$ will usually remove the cross entropy training signal and replace it with the term $\log\left(\sum_{x_1'} p_\theta(x_1'|W_t^-)\mathbb{P}(W_t|W_t^-, x_1')\right)$ which will be used as the training signal for the denoising model $p_\theta(x_1|W_t^-)$. The denoising model is encouraged to be such that the expected jump probability assigns high likelihood to the jump observed under the x_1 conditioned process $\mathbb{Q}^{|x_1}$. This is an indirect training signal for $p_\theta(x_1|W_t^-)$ and one that relies on the arbitrary specification of our $\mathbb{Q}^{|x_1}$ process. We instead show how we can replace this $p_\theta(x_1|W_t^-)$ training signal with the cross entropy loss and be left with a KL term showing that the cross entropy is a lower bound on $\mathcal{L}_{\text{ELBO}}$ minus the rate regularizing term. We summarize this argument in the next section.

Summary To summarize, we have first derived the standard evidence lower bound on the model log-likelihood when using our specific generative rate matrix, $R_t^\theta(x_t, j) = \mathbb{E}_{p_\theta(x_1|x_t)}\left[R_t(x_t, j|x_1)\right]$ for some arbitrarily chosen $R_t(x_t, j|x_1)$ that generates the $p_{t|1}(x_t|x_1)$ conditional flow.

$$
\mathbb{E}_{p_{\text{data}}(x_1)}\left[\log p_\theta(x_1)\right] \geq \mathcal{L}_{\text{ELBO}} + C
$$

We then split $\mathcal{L}_{\text{ELBO}}$ into three terms $\mathcal{L}_{\text{ce}} + \mathcal{L}_R + \mathcal{L}_{\text{KL}}$. We have seen how the term $\mathcal{L}_{\text{KL}}$ allows us to remove the standard $\mathcal{L}_{\text{ELBO}}$ training signal for the denoising model $p_\theta(x_1|x_t)$ and replace it with the cross entropy, creating the $\mathcal{L}_{\text{ce}}$ term. This creates a looser bound if we are to train without the $\mathcal{L}_{\text{KL}}$ term,

$$\mathbb{E}_{p_{\text{data}}(x_1)}\left[\log p_\theta(x_1)\right] \geq \mathcal{L}_{\text{ce}} + \mathcal{L}_R + C$$

We then argue that $\mathcal{L}_R$ is close to $\hat{\mathcal{L}}_R$ which is an unnecessary forcing term encouraging our generative rate to achieve a similar jump rate to our chosen $R_t(x_t, j|x_1)$ even though this R_t matrix is an arbitrary decision and will have a different jump rate depending on which R_t is chosen. We are then left with the standard cross entropy term as our final objective for $p_\theta(x_1|x_t)$ with a final modification to its unweighted form for implementation ease.

C.2.1. Objective for the Masking Interpolant

In this section we will show that $\mathcal{L}_{\text{ELBO}}$ is exactly the weighted cross entropy for the case when we use the masking form for $p_{t|1}(x_t|x_1)$. We note that a similar result has been proven by Austin et al. (2021) for the discrete time diffusion model, and here we verify that this result also holds for our DFM model. We will assume multi-dimensional data, $x_1 \in \{1, \dots, S\}^D$. We refer to Appendix E for the details of the multi-dimensional setting. We will also assume that we use R_t^* as our rate matrix that generates the $p_{t|1}(x_t|x_1)$ conditional flow.

Before we manipulate $\mathcal{L}_{\text{ELBO}}$, we will first find the forms of $R_t^*(x_t^{1:D}, j^{1:D}|x_1^{1:D})$, $R_t^\theta(x_t^{1:D}, j^{1:D})$ and $R_t^\theta(x_t^{1:D})$ for the masking case. From Appendix F.1, equation (31) we have,

$$R_t^{*d}(x_t^d, j^d|x_1^d) = \frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\delta\left\{x_t^d, M\right\}$$

and so

$$\begin{aligned} R_t^*(x_t^{1:D}, j^{1:D}|x_1^{1:D}) &= \sum_{d=1}^D \delta\left\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\right\} R_t^*(x_t^d, j^d|x_1^d) \\ &= \sum_{d=1}^D \delta\left\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\right\}\delta\left\{j^d, x_1^d\right\}\delta\left\{x_t^d, M\right\}\frac{1}{1-t} \end{aligned}$$

From Appendix F.1, equation (32) we have that,

$$R_t^{\theta d}(x_t^{1:D}, j^d) = \frac{p_\theta(x_1^d = j^d|x_t^{1:D})}{1-t}\delta\left\{x_t^d, M\right\}$$

and therefore,

$$\begin{aligned} R_t^\theta(x_t^{1:D}, j^{1:D}) &= \sum_{d=1}^D \delta\left\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\right\} R_t^{\theta d}(x_t^{1:D}, j^d) \\ &= \sum_{d=1}^D \delta\left\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\right\}\frac{p_\theta(x_1^d = j^d|x_t^{1:D})}{1-t}\delta\left\{x_t^d, M\right\} \end{aligned}$$

We now find $R_t^\theta(x_t^{1:D})$

$$\begin{aligned} R_t^\theta(x_t^{1:D}) &= \sum_{j^{1:D}\neq x_t^{1:D}} R_t^\theta(x_t^{1:D}, j^{1:D}) \\ &= \sum_{j^{1:D}}\left(1 - \delta\left\{j^{1:D}, x_t^{1:D}\right\}\right)\sum_{d=1}^D \delta\left\{j^{1:D\setminus d}, x_t^{1:D\setminus d}\right\}\frac{p_\theta(x_1^d = j^d|x_t^{1:D})}{1-t}\delta\left\{x_t^d, M\right\} \\ &= \sum_{d=1}^D \sum_{j^{1:D\setminus d}} \delta\left\{j^{1:D\setminus d}, x_t^{1:D\setminus d}\right\}\delta\left\{x_t^d, M\right\}\frac{1}{1-t}\sum_{j^d}\left(1 - \delta\left\{j^{1:D}, x_t^{1:D}\right\}\right)p_\theta(x_1^d = j^d|x_t^{1:D}) \end{aligned}$$

$$= \sum_{d=1}^{D} \delta\left\{x_t^d, M\right\} \frac{1}{1-t} \sum_{j^d} \left(1 - \delta\left\{j^d, x_t^d\right\}\right) p_\theta(x_1^d = j^d | x_t^{1:D})$$
$$= \sum_{d=1}^{D} \delta\left\{x_t^d, M\right\} \frac{1}{1-t} \sum_{j^d \neq x_t^d} p_\theta(x_1^d = j^d | x_t^{1:D})$$
$$= \sum_{d=1}^{D} \delta\left\{x_t^d, M\right\} \frac{1}{1-t}$$

where on the final line we have used the fact that $p_\theta(x_1^d = M | x_t^{1:D}) = 0$.

We are now ready to manipulate the form of $\mathcal{L}_{\text{ELBO}}$. We start with

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \left(- \int_{t=0}^{t=1} R_t^\theta(W_t) \mathrm{d}t + \sum_{t: W_t^- \neq W_t} \log\left(R_t^\theta(W_t^-, W_t)\right) \right) + C$$

We then apply Dynkin's formula

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \left(\int_{t=0}^{t=1} -R_t^\theta(W_t) + \sum_{y \neq W_t} R_t^*(W_t, y | x_1) \log\left(R_t^\theta(W_t, y)\right) \mathrm{d}t \right) + C$$

We now substitute in the masking forms for $R_t^\theta(W_t)$, $R_t^*(W_t, y|x_1)$ and $R_t^\theta(W_t, y)$

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \Bigg(\int_{t=0}^{t=1} \left(-\sum_{d=1}^{D} \delta\left\{W_t^d, M\right\} \frac{1}{1-t} \right) + \sum_{y^{1:D} \neq W_t^{1:D}} \left\{ \left(\sum_{d=1}^{D} \delta\left\{W_t^{1:D\setminus d}, y^{1:D\setminus d}\right\} \delta\left\{y^d, x_1^d\right\} \delta\left\{W_t^d, M\right\} \frac{1}{1-t} \right) \times \log\left(\sum_{d=1}^{D} \delta\left\{W_t^{1:D\setminus d}, y^{1:D\setminus d}\right\} \delta\left\{W_t^d, M\right\} p_\theta(y^d | W_t^{1:D}) \frac{1}{1-t} \right) \right\} \mathrm{d}t \Bigg) + C$$

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \left(\int_{t=0}^{t=1} \sum_{d=1}^{D} \sum_{y^d \neq W_t^d} \delta\left\{W_t^d, M\right\} \delta\left\{y^d, x_1^d\right\} \frac{1}{1-t} \log\left(p_\theta(y^d | W_t^{1:D})\right) \mathrm{d}t \right) + C$$

where we have moved terms that don't depend on θ into the constant.

$$\mathcal{L}_{\text{ELBO}} = \int p_{\text{data}}(\mathrm{d}x_1) \mathbb{Q}^{|x_1}(\mathrm{d}\omega) \left(\int_{t=0}^{t=1} \sum_{d=1}^{D} \delta\left\{W_t^d, M\right\} \frac{1}{1-t} \log\left(p_\theta(x_1^d | W_t^{1:D})\right) \mathrm{d}t \right) + C$$
$$= \mathbb{E}_{\mathcal{U}(t;0,1) p_{\text{data}}(x_1) p_t(x_t|x_1)} \left[\sum_{d=1}^{D} \delta\left\{x_t^d, M\right\} \frac{1}{1-t} \log p_\theta(x_1^d | x_t^{1:D}) \right] + C \tag{24}$$

where we have arrived at the weighted cross entropy, weighted by $\frac{1}{1-t}$ and only calculated for dimensions that are masked in our corrupted sample x_t.

D. Discussion of Related Work

Flow based methods for generative modelling were introduced by (Liu et al., 2023; Albergo & Vanden-Eijnden, 2023; Lipman et al., 2023). These methods simplify the generative modelling framework over diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020) by considering noise-data interpolants rather than considering forward/backward

diffusions. This work brings these benefits to discrete data denoising models which previously have used the diffusion methodology (Sohl-Dickstein et al., 2015; Hoogeboom et al., 2021; Austin et al., 2021) relying on forward/backward processes defined by Markov transition kernels. Specifically, prior discrete diffusion works first define a forward noising process with a rate matrix $\tilde{R}_t$. This defines infinitesimal noise additions. To train the model, we need access to the equivalent of $p_{t|1}$, i.e. the total amount of noise added simulating from 1 to t. To find this value, the matrix exponential needs to be applied to the forward rate matrix, $p_{t|1} = \exp\left(\int_t^1 \tilde{R}_s \mathrm{d}s\right)$. This means that discrete diffusion models are limited in the choice of forward noising process. The choice of $\tilde{R}_t$ must be such that the matrix exponential is tractable. For DFM, we simply write down $p_{t|1}$ rather than implicitly defining it through the matrix exponential and then can find a rate matrix to simulate with by differentiating $p_{t|1}$ and using R_t^*. Furthermore, the standard ELBO objective used to train discrete diffusion models depends on the initial choice of $\tilde{R}_t$. At sample time, it is then standard to simulate with the time reversal of $\tilde{R}_t$. This needlessly limits the choice of simulation process as we have shown in this work that there are infinitely many valid choices of rate matrices that could be used for sampling.

There have been post-hoc changes to the sampling process made in prior work e.g. corrector steps used by Campbell et al. (2022), however due to the ELBO maximizing the model log-likelihood under the assumption of sampling using the time-reversal, the diffusion framework still revolves around one 'canonical' sample time process (the time-reversal) whereas DFM makes it clear this choice is arbitrary and the sample process can be chosen at inference time for best performance.

Previous discrete diffusion works have also suggested alternatives to the ELBO. Sun et al. (2023b) introduce a categorical score matching loss that resembles the cross entropy, however, the denoising network is required to make a prediction x_0^d based only on the other $D-1$ dimensions of the input noisy state, $x_t^{1:D\setminus d}$. This requires specialized architectures and methods to remain computationally efficient. Vignac et al. (2023a) propose to learn a diffusion based model solely using the cross-entropy but do not analyse the link between the cross-entropy and the log-likelihood of the model as we do in App. C. Meng et al. (2022) propose to learn a discrete score model based on data ratios using an L2 based loss which has some undesirable properties such as not penalizing mode dropping as described by Lou et al. (2023). Lou et al. (2023) refine this approach and propose to learn data ratios using the score entropy loss which, like the standard cross entropy, does not depend on the choice of forward rate matrix. However, in order for the score entropy to be a true ELBO, the forward rate matrix needs to be used as a weighting factor.

Multimodal diffusion models have been applied to tabular data (Kotelnikov et al., 2023) where continuous diffusion is used for continuous features and a uniform style of corruption under a discrete diffusion framework is applied to discrete features. This idea was then expanded to molecule generation where the task is to generate a molecules atom types, their positions and their connectivity. Peng et al. (2023) use a masking process for the discrete atom types and bond types with a continuous space process for the atom positions. Vignac et al. (2023b) use a discrete process converging towards the independent marginal distribution in each dimension (Vignac et al., 2023a) for atom types, bond types and formal charges of the molecules along with a continuous process for atom positions. Hua et al. (2023) use a uniform discrete process for bond types with a continuous space process applied to atom positions as well as atom features embedded in continuous space. These works also investigate the importance of the multimodal noise schedule. Peng et al. (2023) find that corrupting the bonds first and then the atom positions improves performance by avoiding unphysical bonds appearing in the corruption process. Vignac et al. (2023b) have a similar finding that during corruption, the atom types should be corrupted first, then the bond types and finally the atom positions. We generalize these ideas by using the approach of Albergo et al. (2023) and learning our model over all relative levels of noise between our modalities. This allows picking the desired path through the multimodal noise landscape at inference time either performing co-generation, inverse folding or forward folding.

Other approaches for discrete data modelling opt to embed the data into a continuous space in order to still use the continuous diffusion framework (Li et al., 2022; Chen et al., 2023a; Richemond et al., 2022; Gong et al., 2023; Dieleman et al., 2022; Han et al., 2022; Strudel et al., 2022; Gulrajani & Hashimoto, 2023; Floto et al., 2023), however, this loses the discrete structure of the data during generation. This can be important when the quantity that is represented by the discrete variable as algorithmic importance. For example, Qin et al. (2023) perform sparse graph generation where the discrete token represents the existence of an edge. It is then important for the edge to be known to physically exist or not so that sparse graph networks can be applied to the problem.

General Fokker-Planck equations on discrete state spaces (Chow et al., 2012) have been used to construct sampling methods for energy functions (Sun et al., 2023a). Further, in a generative modelling context, the Kolmogorov equation has been used to construct equivalent diffusion processes with fewer transitions (Zhang et al., 2023) making links to optimal transport. We

take this idea further to build a generative modelling paradigm around the flexibility of the Kolmogorov equation.

The construction of discrete diffusion model from a marginal distribution perspective as opposed to a forward corruption process has also been used by Chen et al. (2023b). Their method defines the marginal distribution at time t as a combination of the data and a noise sample and then finds a process that generates those marginals, for the masking and uniform case. They use this to create a faster sampling algorithm by exploiting the fact that if you have a low stochasticity process, you know there should only be D transitions in the masking case (although this is not the case in the unconditional uniform case). Therefore, when conditioning on these times, only D function evaluations are needed. This approach could also be used with a DFM when $\eta = 0$, however, our general framework also demonstrates the benefits of allowing $\eta > 0$.

The consideration of flows on discrete state spaces has also been used to construct GFlowNet algorithms (Bengio et al., 2023) which aim to sample from a given energy function. Here we instead focus on the the generative modeling context where we aim to sample novel datapoints when only given access to some dataset of training examples. GFlowNets also can use the detailed balance equation Eq. (9) as a training training objective. Detailed balance is also used in Markov Chain Monte Carlo methods (Metropolis et al., 1953; Hastings, 1970) to construct a transition probability with the desired energy function that we wish to sample from as its stationary distribution. In our work, we use the detailed balance condition as a way to increase the inference time flexibility in our framework

E. Multidimensional Data

In this section we derive how we can efficiently model D dimensional data, $x_1 \in \{1, \dots, S\}^D$ by using factorization assumptions. When we wish to emphasize the multidimensional aspect we can write $x_1^{1:D}$ and use $x_1^d \in \{1, \dots, S\}$ to refer to the value in dimension d. We use $1:D \backslash d$ to denote all dimensions except d. To operate in multidimensional spaces, we will make the following assumptions

- **Assumption 1** $p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \prod_{d=1}^D p_{t|1}(x_t^d|x_1^d)$
- **Assumption 2** $p_{t|1}(x_t^d|x_1^d) = 0 \implies \partial_t p_{t|1}(x_t^d|x_1^d) = 0, \forall d$
- **Assumption 3** $R_t(x_t^{1:D}, j^{1:D}|x_1^{1:D}) = \sum_{d=1}^D \delta\{x_t^{1:D\backslash d}, j^{1:D\backslash d}\} R_t^d(x_t^d, j^d|x_1^d)$

The first assumption creates independent corruption processes in each dimension, similar to the factorization assumptions made in diffusion models where the forward noising processes proceed independently in each dimension. Assumption 2 is the same assumption we made in order to derive R_t^* in 1-dimension but now we assume it individually for every dimension. Finally, assumption 3 states that for our data conditional rate matrix, it decomposes into a sum of rate matrices for each dimension and so the rate for transitions that change more than 1 dimension at a time are 0. This is the same assumption made by Campbell et al. (2022) in order to make calculations tractable. We will enable our process to make multiple dimensional changes simultaneously later when we come to derive our sampling algorithm.

Under these assumptions, we will now derive DFM for the multidimensional case. We start with the data conditional Kolmogorov equation

$$\partial_t p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \sum_{j^{1:D}} R_t(j^{1:D}, x_t^{1:D}|x_1^{1:D}) p_{t|1}(j^{1:D}|x_t^{1:D}) \tag{25}$$

We now substitute the form for the rate matrix under Assumption 3 into the RHS of (25) to get

$$\begin{aligned} \text{RHS} &= \sum_{j^{1:D}} \sum_{d=1}^D \delta\{x_t^{1:D\backslash d}, j^{1:D\backslash d}\} R_t^d(j^d, x_t^d|x_1^d) p_{t|1}(j^{1:D}|x_1^{1:D}) \\ &= \sum_{d=1}^D \sum_{j^d} R_t^d(j^d, x_t^d|x_1^d) p_{t|1}(x_t^{1:D\backslash d} \odot j^d|x_1^{1:D}) \end{aligned} \tag{26}$$

where we use $x_t^{1:D\backslash d} \odot j^d$ to denote a vector of dimension D where in the d-th dimension it has the value of j^d and in the other dimensions it has values $x_t^{1:D\backslash d}$. We now verify that the following form for R_t^d satisfies the Kolmogorov equation,

$$R_t^{*d}(x_t^d, j^d|x_1^d) = \begin{cases} \frac{\text{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right)}{Z_t^d p_{t|1}(x_t^d|x_1^d)} & \text{for } p_{t|1}(x_t^d|x_1^d) > 0, p_{t|1}(j^d|x_1^d) > 0 \\ = 0 & \text{otherwise} \end{cases} \tag{27}$$

where $\mathcal{Z}_t^d = |\{j^d : p_{t|1}(j^d|x_1^d) > 0\}|$ and we only define R_t^{*d} for off-diagonal entries, $x_t^d \neq j^d$ remembering that $R_t^{*d}(x_t^d, x_t^d|x_1^d) = -\sum_{j^d \neq x_t^d} R_t^{*d}(x_t^d, j^d|x_1^d)$.

We first assume $p_{t|1}(x_t^d|x_1^d) > 0 \, \forall d$ and substitute in R_t^{*d} into equation (26).

$$\text{RHS} = \sum_{d=1}^{D} \sum_{j^d \neq x_t^d, p_{t|1}(j^d|x_1^d)>0} \left(\frac{\text{ReLU}\left(\partial_t p_{t|1}(x_t^d|x_1^d) - \partial_t p_{t|1}(j^d|x_1^d)\right)}{\mathcal{Z}_t^d p_{t|1}(j^d|x_1^d)} p_{t|1}(x_t^{1:D\setminus d} \odot j^d|x_1^{1:D}) \right.$$
$$\left. - \frac{\text{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right)}{\mathcal{Z}_t^d p_{t|1}(x_t^d|x_1^d)} p_{t|1}(x_t^{1:D}|x_1^{1:D}) \right)$$

$$\text{RHS} = \sum_{d=1}^{D} \frac{1}{\mathcal{Z}_t^d} p_{t|1}(x_t^{1:D\setminus d}|x_1^{1:D}) \sum_{j^d \neq i^d, p_{t|1}(j^d|x_1^d)>0} \left(\text{ReLU}\left(\partial_t p_{t|1}(x_t^d|x_1^d) - \partial_t p_{t|1}(j^d|x_1^d)\right) \right.$$
$$\left. - \text{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right) \right)$$

$$\begin{aligned}
\text{RHS} &= \sum_{d=1}^{D} \frac{1}{\mathcal{Z}_t^d} p_{t|1}(x_t^{1:D\setminus d}|x_1^{1:D}) \sum_{j^d \neq i^d, p_{t|1}(j^d|x_1^d)>0} \left(\partial_t p_{t|1}(x_t^d|x_1^d) - \partial_t p_{t|1}(j^d|x_1^d) \right) \\
&= \sum_{d=1}^{D} p_{t|1}(x_t^{1:D\setminus d}|x_1^{1:D}) \partial_t p_{t|1}(x_t^d|x_1^d) \\
&= \partial_t \left(\prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d) \right) \\
&= \text{LHS}
\end{aligned}$$

where we have used the fact that $p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d)$.

For the case that there exists a d' for which $p_{t|1}(x_t^{d'}|x_1^{d'}) = 0$ we have $\partial_t p_{t|1}(x_t^{d'}|x_1^{d'}) = 0$ by assumption. We first examine the LHS of equation (25) in this case.

$$\begin{aligned}
\text{LHS} &= \partial_t p_{t|1}(x_t^{1:D}|x_1^{1:D}) \\
&= \partial_t \left(\prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d) \right) \\
&= \sum_{d=1}^{D} p_{t|1}(x_t^{1:D\setminus d}|x_1^{1:D\setminus d}) \partial_t p_{t|1}(x_t^d|x_1^d) \\
&= p_{t|1}(x_t^{1:D\setminus d'}|x_1^{1:D\setminus d'}) \partial_t p_{t|1}(x_t^{d'}|x_1^{d'}) + \sum_{d=1\setminus d'}^{D} p_{t|1}(x_t^{1:D\setminus d}|x_1^{1:D\setminus d}) \partial_t p_{t|1}(x_t^d|x_1^d) \\
&= p_{t|1}(x_t^{1:D\setminus d'}|x_1^{1:D\setminus d'}) \underbrace{\partial_t p_{t|1}(x_t^{d'}|x_1^{d'})}_{0} + \sum_{d=1\setminus d'}^{D} \underbrace{p_{t|1}(x_t^{d'}|x_1^{d'})}_{0} p_{t|1}(x_t^{1:D\setminus d,d'}|x_1^{1:D\setminus d,d'}) \partial_t p_{t|1}(x_t^d|x_1^d) \\
&= 0
\end{aligned}$$

where we use $1:D\setminus d, d'$ to mean all dimensions except d and d'. We now examine the RHS of equation (25).

$$\text{RHS} = \sum_{d=1}^{D} \sum_{j^d} R_t^{*d}(j^d, x_t^d|x_1^d) p_{t|1}(x_t^{1:D\setminus d} \odot j^d|x_1^{1:D})$$

$$
\begin{aligned}
&= \sum_{j^{d'}} R_t^{*d'}(j^{d'}, x_t^{d'}|x_1^{d'}) p_{t|1}(x_t^{1:D\backslash d'} \odot j^{d'}|x_1^{1:D}) + \sum_{d=1\backslash d'}^{D} \sum_{j^d} R_t^{*d}(j^d, x_t^d|x_1^d) p_{t|1}(x_t^{1:D\backslash d} \odot j^d|x_1^{1:D}) \\
&= \sum_{j^{d'}} \underbrace{R_t^{*d'}(j^{d'}, x_t^{d'}|x_1^{d'})}_{0} p_{t|1}(x_t^{1:D\backslash d'} \odot j^{d'}|x_1^{1:D}) \\
&\quad + \sum_{d=1\backslash d'}^{D} \sum_{j^d} R_t^{*d}(j^d, x_t^d|x_1^d) \underbrace{p_{t|1}(x_t^{d'}|x_1^{d'})}_{0} p_{t|1}(x_t^{1:D\backslash d,d'} \odot j^d|x_1^{1:D\backslash d'}) \\
&= 0 \\
&= \text{LHS}
\end{aligned}
$$

where we have used the fact that $R_t^{*d'}(j^{d'}, x_t^{d'}|x_1^{d'}) = 0$ because $p_{t|1}(j^{d'}|x_1^{d'}) = 0$ Therefore, for both cases we have R_t^* satisfies the conditional Kolmogorov equation (25) and thus we have found a rate matrix that generates our desired conditional flow. The final step is to convert this rate matrix conditioned on $x_1^{1:D}$ into an unconditional rate matrix that can be used for generative modeling. We first write down the unconditional multi-dimensional Kolmogorov equation

$$
\partial_t p_t(x_t^{1:D}) = \sum_{j^{1:D}} R_t(j^{1:D}, x_t^{1:D}) p_t(j^{1:D}) \tag{28}
$$

We now make the following assumption for the form of the unconditional rate matrix and verify that it indeed satisfies the unconditional multi-dimensional Kolmogorov equation, (28).

$$
R_t(x_t^{1:D}, j^{1:D}) = \sum_{d=1}^{D} \delta\{x_t^{1:D\backslash d}, j^{1:D\backslash d}\} R_t^d(x_t^{1:D}, j^d) \tag{29}
$$

with

$$
R_t^d(x_t^{1:D}, j^d) = \mathbb{E}_{p(x_1^d|x_t^{1:D})}\left[R_t^{*d}(x_t^d, j^d|x_1^d)\right]
$$

with $R_t^{*d}(x_t^d, j^d|x_1^d)$ being given by (27). Substitute this form into (28)

$$
\begin{aligned}
\text{RHS} &= \sum_{j^{1:D}} \sum_{d=1}^{D} \delta\{j^{1:D\backslash d}, x_t^{1:D\backslash d}\} \mathbb{E}_{p(x_1^d|j^{1:D})}\left[R_t^{*d}(j^d, x_t^d|x_1^d)\right] p_t(j^{1:D}) \\
&= \sum_{d=1}^{D} \sum_{j^d} \mathbb{E}_{p(x_1^d|x_t^{1:D\backslash d} \odot j^d)}\left[R_t^{*d}(j^d, x_t^d|x_1^d)\right] p_t(x_t^{1:D\backslash d} \odot j^d) \\
&= \sum_{d=1}^{D} \sum_{j^d} \sum_{x_1^d} p(x_1^d|x_t^{1:D\backslash d} \odot j^d) R_t^{*d}(j^d, x_t^d|x_1^d) p_t(x_t^{1:D\backslash d} \odot j^d) \\
&= \sum_{d=1}^{D} \sum_{j^d} \sum_{x_1^d} p(x_1^d|x_t^{1:D\backslash d} \odot j^d) R_t^{*d}(j^d, x_t^d|x_1^d) p_t(x_t^{1:D\backslash d} \odot j^d) \underbrace{\sum_{x_1^{1:D\backslash d}} p(x_1^{1:D\backslash d}|x_1^d, x_t^{1:D\backslash d} \odot j^d)}_{=1} \\
&= \sum_{d=1}^{D} \sum_{j^d} \sum_{x_1^{1:D}} p(x_1^{1:D}|x_t^{1:D\backslash d} \odot j^d) R_t^{*d}(j^d, x_t^d|x_1^d) p_t(x_t^{1:D\backslash d} \odot j^d) \\
&= \sum_{d=1}^{D} \sum_{j^d} \sum_{x_1^{1:D}} p_{\text{data}}(x_1^{1:D}) p_{t|1}(x_t^{1:D\backslash d} \odot j^d|x_1^{1:D}) R_t^{*d}(j^d, x_t^d|x_1^d) \\
&= \mathbb{E}_{p_{\text{data}}(x_1^{1:D})}\left[\sum_{d=1}^{D} \sum_{j^d} p_{t|1}(x_t^{1:D\backslash d} \odot j^d|x_1^{1:D}) R_t^{*d}(j^d, x_t^d|x_1^d)\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \mathbb{E}_{p_{\text{data}}(x_1^{1:D})}\left[\partial_t p_{t|1}(x_t^{1:D}|x_1^{1:D})\right] \quad \text{by (26)} \\
&= \partial_t p_t(x_t^{1:D}) \\
&= \text{LHS}
\end{aligned}
$$

where we have used Eq. (26) with the fact that we know R_t^* given by Eq. (27) satisfies the conditional Kolmogorov equation Eq. (25). We have now verified that the rate given by Eq. (29) gives us our desired unconditional flow and we can use it for generative modeling.

E.1. Training

In order to approximate the true generative rate matrix given by equation (29), we need approximations to the denoising distributions in each dimension, $p(x_1^d|x_t^{1:D})$, for $d = 1, \dots, D$. We can parameterize these conditionally independent x_1^d distributions through a neural network that outputs logits of shape $D \times S$ when given input $x_t^{1:D}$ of shape D. We then apply a softmax to the logits to obtain approximate denoising probabilities $p_\theta(x_1^d|x_t^{1:D})$, $d = 1, \dots, D$ of shape $D \times S$. We learn the parameters of the neural network with the cross entropy loss for each dimension

$$
\mathcal{L}_{\text{ce}} = \mathbb{E}_{p_{\text{data}}(x_1^{1:D})\mathcal{U}(t;0,1)p_{t|1}(x_t^{1:D}|x_1^{1:D})}\left[\sum_{d=1}^{D} \log p_{1|t}^\theta(x_1^d|x_t^{1:D})\right]
$$

E.2. Sampling

The standard Euler step transition probability for our CTMC defined through our learned denoising model with time step Δt is

$$
\begin{aligned}
p_{t+\Delta t|t}(j^{1:D}|x_t^{1:D}) &= \delta\{x_t^{1:D}, j^{1:D}\} + R_t^\theta(x_t^{1:D}, j^{1:D})\Delta t \\
&= \delta\{x_t^{1:D}, j^{1:D}\} + \sum_{d=1}^{D} \delta\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\}\mathbb{E}_{p_\theta(x_1^d|x_t^{1:D})}\left[R_t^d(x_t^d, j^d|x_1^d)\right]\Delta t
\end{aligned} \tag{30}
$$

In this form, we would be unable to make transition steps that involve more than 1 dimension changing at a time due to our factorized form for $R_t^\theta(x_t^{1:D}, j^{1:D})$. To enable multiple dimensions to transition simultaneously in a single update step we can approximate the standard Euler transition step (30) with a factorized version $\tilde{p}_{t+\Delta t|t}(j^{1:D}|x_t^{1:D})$ with the following form

$$
\begin{aligned}
\tilde{p}_{t+\Delta t|t}(j^{1:D}|x_t^{1:D}) &= \prod_{d=1}^{D} \tilde{p}_{t+\Delta t|t}^d(j^d|x_t^{1:D}) \\
&= \prod_{d=1}^{D} \left\{\delta\{x_t^d, j^d\} + \mathbb{E}_{p_\theta(x_1^d|x_t^{1:D})}\left[R_t^d(x_t^d, j^d|x_1^d)\right]\Delta t\right\} \\
&= \delta\{x_t^{1:D}, j^{1:D}\} + \sum_{d=1}^{D} \delta\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\}\mathbb{E}_{p_\theta(x_1^d|x_t^{1:D})}\left[R_t^d(x_t^d, j^d|x_1^d)\right]\Delta t + O(\Delta t^2)
\end{aligned}
$$

where we can see on the final line that $\tilde{p}_{t+\Delta t|t}$ approximates $p_{t+\Delta t|t}$ to first order. Sampling from $\tilde{p}_{t+\Delta t|t}$ can be seen as taking an Euler step in each dimension independently for each simulation step.

We note this sampling method is similar to the tau-leaping method used in prior CTMC based approaches (Gillespie, 2001; Campbell et al., 2022) however tau-leaping allows multiple jumps to be made in the same dimensions which is unsuitable for categorical data.

E.3. Detailed Balance

In this section we verify that if we achieve detailed balance individually and independently in each dimension, then our full dimensional process will also be in detailed balance.

Consider the multidimensional detailed balance equation

$$
p_{t|1}(x_t^{1:D}|x_1^{1:D})R_t(x_t^{1:D}, j^{1:D}|x_1^{1:D}) = p_{t|1}(j^{1:D}|x_1^{1:D})R_t(j^{1:D}, x_t^{1:D}|x_1^{1:D})
$$

Now, substitute in our factorized forms for $R_t(x_t^{1:D}, j^{1:D}|x_1^{1:D})$ and $p_{t|1}(x_t^{1:D}|x_1^{1:D})$

$$\left(\prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d)\right)\left(\sum_{d=1}^{D}\delta\{x_t^{1:D\setminus d}, j^{1:D\setminus d}\}R_t^d(x_t^d, j^d|x_1^{d)})\right) = \left(\prod_{d=1}^{D} p_{t|1}(j^d|x_1^d)\right)\left(\sum_{d=1}^{D}\delta\{j^{1:D\setminus d}, x_t^{1:D\setminus d}\}R_t^d(j^d, x_t^d|x_1^d)\right)$$

Now, both sides are 0 for when x_t and j differ in more than one dimension. Consider the case when they differ in exactly one dimension, call it d. The detailed balance equation simplifies to

$$p_{t|1}(x_t^d|x_1^d)R_t^d(x_t^d, j^d|x_1^d) = p_{t|1}(j^d|x_1^d)R_t^d(j^d, x_t^d|x_1^d)$$

which we note is the standard single dimensional detailed balance equation for dimension d. Therefore, if our R_t^d matrices are all in detailed balance with their respective $p_{t|1}(x_t^d|x_1^d)$ conditional marginals, then the full dimensional rate matrix $R_t(x_t^{1:D}, j^{1:D}|x_1^{1:D})$ will also be in detailed balance with the full dimensional conditional marginals $p_{t|1}(x_t^{1:D}|x_1^{1:D})$.

F. Implementation Details

In this section we provide concrete derivations of our DFM method. We use a masking process in App. F.1, a uniform process in App. F.2 and explore the general case for any given $p_{t|1}$ in App. F.3. We also provide minimal PyTorch implementations for our training and sampling loops in each case. We will assume multi-dimensional data under the factorization assumptions listed in App. E.

Notebooks containing these minimal examples can be found at `https://github.com/andrew-cr/discrete_flow_models`.

F.1. Masking Example

Here, we assume the masking form for $p_{t|1}$. We begin by writing this data conditional flow

$$\begin{aligned} p_{t|1}(x_t^{1:D}|x_1^{1:D}) &= \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d) \\ &= \prod_{d=1}^{D}\left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\delta\left\{x_t^d, M\right\}\right) \end{aligned}$$

This is the distribution we will use to train our denoising model $p_{1|t}^{\theta}(x_1^{1:D}|x_t^{1:D})$. PyTorch code for the training loop is given in Listing 1

Listing 1. Masking Training loop

```
import torch
import torch.nn.functional as F


# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S-1). We know
    the clean data contains no masks and hence we only need to output logits
    over the valid values.

mask_index = S - 1 # Assume the final state is the mask state

for x1 in dataset:
    # x1 has shape (B, D)
    optimizer.zero_grad()
    t = torch.rand((B,))
    xt = x1.clone()
    xt[torch.rand((B,D)) < (1 - t[:, None])] = mask_index
    logits = model(xt, t) # (B, D, S-1)
    x1[xt != mask_index] = -1 # Don't compute the loss on unmasked dimensions
    loss = F.cross_entropy(logits.transpose(1, 2), x1, reduction='mean',
        ignore_index=-1)
    loss.backward()
    optimizer.step()
```

We will also derive the form for $R_t^{*d}(i^d, j^d|x_1^d)$. For this we need to find $\partial_t p_{t|1}(x_t^d|x_1^d)$.

$$\begin{aligned}\partial_t p_{t|1}(x_t^d|x_1^d) &= \partial_t \left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\delta\left\{x_t^d, M\right\}\right) \\ &= \delta\left\{x_t^d, x_1^d\right\} - \delta\left\{x_t^d, M\right\}\end{aligned}$$

We can now find $R_t^{*d}(x_t^d, j^d|x_1^d)$. When working with rate matrices in this section, we will always assume $x_t^d \neq j^d$ and calculate the diagonal entries as $R_t(i,i) = -\sum_{j\neq i} R_t(i,j)$ later. We note that $R_t^{*d}(x_t^d, j^d|x_1^d) = 0$ for $p_{t|1}(x_t^d|x_1^d) = 0$ or $p_{t|1}(j^d|x_1^d) = 0$. Further, our initial distribution $p_0(x_0^{1:D}) = \prod_{d=1}^D \delta\left\{x_0^d, M\right\}$. Therefore, at all points in our CTMC, x_t^d is only ever M or x_1^d. Furthermore, we only ever have to consider transitions to a j^d that is either $j^d = M$ or $j^d = x_1^d$. Now, for $p_{t|1}(x_t^d|x_1^{1:D}) > 0$ and $p_{t|1}(j^d|x_1^{1:D}) > 0$ we have

$$\begin{aligned}R_t^{*d}(x_t^d, j^d|x_1^d) &= \frac{\text{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right)}{\mathcal{Z}_t^d p_{t|1}(x_t^d|x_1^d)} \\ &= \frac{\text{ReLU}\left(\delta\left\{j^d, x_1^d\right\} - \delta\left\{j^d, M\right\} - \delta\left\{x_t^d, x_1^d\right\} + \delta\left\{x_t^d, M\right\}\right)}{2\left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\delta\left\{x_t^d, M\right\}\right)} \\ &= \frac{1}{1-t} \quad \text{for } j^d = x_1^d, x_t^d = M \text{ and 0 otherwise}\end{aligned} \tag{31}$$

We note here that our calculation may not strictly be valid for exactly $t = 0$ or $t = 1$ but are valid for any $t \in (0, 1)$ and so we can simply ignore these edge cases, see App. B.2 for further discussion. Now we find our unconditional rate matrix

$$\begin{aligned}R_t^{\theta d}(x_t^{1:D}, j^d) &= \mathbb{E}_{p^\theta_{1|t}(x_1^d|x_t^{1:D})}\left[R_t^{*d}(x_t^d, j^d|x_1^d)\right] \\ &= \mathbb{E}_{p^\theta_{1|t}(x_1^d|x_t^{1:D})}\left[\frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\delta\left\{x_t^d, M\right\}\right] \\ &= \frac{p^\theta_{1|t}(x_1^d = j^d|x_t^{1:D})}{1-t}\delta\left\{x_t^d, M\right\}\end{aligned} \tag{32}$$

Our transition step is then

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \delta\left\{j^d, x_t^d\right\} + R_t^{\theta d}(x_t^{1:D}, j^d)\Delta t$$

For $j^d \neq x_t^d$ this is

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \Delta t \frac{p_{1|t}^\theta(x_1^d = j^d|x_t^{1:D})}{1-t} \delta\left\{x_t^d, M\right\} \tag{33}$$

For $j^d = x_t^d$ this is

$$\begin{aligned} p_{t+\Delta t|t}(j^d = x_t^d|x_t^{1:D}) &= 1 - \sum_{k \neq x_t^d} p_{t+\Delta t|t}(k|x_t^{1:D}) \\ &= 1 - \sum_{k \neq x_t^d} \Delta t \frac{p_{1|t}^\theta(x_1^d = k|x_t^{1:D})}{1-t} \delta\left\{x_t^d, M\right\} \\ &= 1 - \frac{\Delta t}{1-t} \delta\left\{x_t^d, M\right\} \end{aligned}$$

where on the final line we have used the fact that when $p_{1|t}^\theta(x_1^d = M|x_t^{1:D}) = 0$. Therefore, if $x_t^d = M$ then we have a $\frac{\mathrm{d}t}{1-t}$ chance of flipping to some unmasked state with the probabilities for the token to unmask to given by $p_{1|t}^\theta(x_1^d|x_t^{1:D})$. If $x_t^d \neq M$ (i.e. it has already been unmasked) then we simply stay in the current unmasked state.

Listing 2 shows PyTorch code that implements this sampling loop.

Listing 2. Masking Sampling loop

```
import torch
import torch.nn.functional as F
from torch.distributions.categorical import Categorical


# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S-1). We know
    the clean data contains no masks and hence we only need to output logits
    over the valid values.
t = 0.0
dt = 0.001
mask_index = S-1

xt = mask_index * torch.ones((B, D), dtype=torch.long)

while t < 1.0:
    logits = model(xt, t * torch.ones((B,))) # (B, D, S-1)
    x1_probs = F.softmax(logits, dim=-1) # (B, D, S-1)
    x1 = Categorical(x1_probs).sample() # (B, D)
    will_unmask = torch.rand((B, D)) < (dt / (1-t)) # (B, D)
    will_unmask = will_unmask * (xt == mask_index) # (B,D) only unmask currently
        masked positions
    xt[will_unmask] = x1[will_unmask]

    t += dt
```

F.1.1. Detailed Balance

In order to expand our family of rate matrices that we can use at sampling time, we want to find a detailed balance rate matrix R_t^{DB} that satisfies the detailed balance equation

$$p_{t|1}(i|x_1) R_t^{\mathrm{DB}}(i, j|x_1) = p_{t|1}(j|x_1) R_t^{\mathrm{DB}}(j, i|x_1)$$

We now have to make some assumptions on the form for R_t^{DB}. With this masking noise a process that is in detailed balance will have some rate for transitions going from a mask state towards x_1 and some rate for transitions going from x_1 back

towards the mask state. Such a rate would have the following form

$$R_t^{\mathrm{DB}}(i, j|x_1) = a_t\delta\left\{i, x_1\right\}\delta\left\{j, M\right\} + b_t\delta\left\{i, M\right\}\delta\left\{j, x_1\right\}$$

for some constants a_t and b_t that we must find. Substituting this into the detailed balance equation along with the masking interpolation form for $p_{t|1}(x_t|x_1)$ gives

$$\begin{aligned}&\left(t\delta\left\{i, x_1\right\} + (1-t)\delta\left\{i, M\right\}\right)\left(a_t\delta\left\{i, x_1\right\}\delta\left\{j, M\right\} + b_t\delta\left\{i, M\right\}\delta\left\{j, x_1\right\}\right) = \\ &\left(t\delta\left\{j, x_1\right\} + (1-t)\delta\left\{j, M\right\}\right)\left(a_t\delta\left\{j, x_1\right\}\delta\left\{i, M\right\} + b_t\delta\left\{j, M\right\}\delta\left\{i, x_1\right\}\right)\end{aligned}$$

$$ta_t\delta\left\{i, x_1\right\}\delta\left\{j, M\right\} + (1-t)b_t\delta\left\{i, M\right\}\delta\left\{j, x_1\right\} = ta_t\delta\left\{j, x_1\right\}\delta\left\{i, M\right\} + (1-t)b_t\delta\left\{j, M\right\}\delta\left\{i, x_1\right\}$$

This equation must be true for all i, j. Pick $i = x_1$ and $j = M$ to get

$$ta_t = (1-t)b_t$$

If we pick $i = M$ and $j = x_1$ then we would obtain the same equation and if we pick any other values for i, j with $i \neq j$ then we would get $0 = 0$. Note that we will find R_t^{DB} for $i \neq j$ and then the value for $R_t^{\mathrm{DB}}(i, i)$ is simply calculated using $R_t^{\mathrm{DB}}(i, i) = -\sum_{j\neq i} R_t^{\mathrm{DB}}(i, j)$. Since we will obtain no more constraints on the values of a_t and b_t, we will need to pick a value for one of them. We can simply set $a_t = \eta$ where η is our stochasticity parameter since this value sets the rate at which points that are already at x_1 will come off x_1 and travel back to the mask state. This gives $b_t = \frac{\eta t}{1-t}$ and so for $i \neq j$,

$$R_t^{\mathrm{DB}}(i, j|x_1) = \eta\delta\left\{i, x_1\right\}\delta\left\{j, M\right\} + \frac{\eta t}{1-t}\delta\left\{i, M\right\}\delta\left\{j, x_1\right\}.$$

We now combine this rate with R_t^{*d} that we calculated previously to find a new unconditional rate matrix with a variable amount of stochasticity.

$$\begin{aligned}R_t^{\theta d}(x_t^{1:D}, j^d) &= \mathbb{E}_{p_{1|t}^{\theta}(x_1^d|x_t^{1:D})}\left[R_t^{*d}(x_t^d, j^d|x_1^d) + {R^{\mathrm{DB}}}_t^d(x_t^d, j^d|x_1^d)\right] \\ &= \mathbb{E}_{p_{1|t}^{\theta}(x_1^d|x_t^{1:D})}\left[\frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\delta\left\{x_t^d, M\right\} + \eta\delta\left\{x_t^d, x_1^d\right\}\delta\left\{j^d, M\right\} + \frac{\eta t}{1-t}\delta\left\{x_t^d, M\right\}\delta\left\{j^d, x_1^d\right\}\right] \\ &= \frac{p_{1|t}^{\theta}(x_1^d = j^d|x_t^{1:D})}{1-t}\delta\left\{x_t^d, M\right\} + \eta p_{1|t}^{\theta}(x_1^d = x_t^d|x_t^{1:D})\delta\left\{j^d, M\right\} + \frac{\eta t}{1-t}\delta\left\{x_t^d, M\right\}p_{1|t}^{\theta}(x_1^d = j^d|x_t^{1:D}) \\ &= \frac{1+\eta t}{1-t}p_{1|t}^{\theta}(x_1^d = j^d|x_t^{1:D})\delta\left\{x_t^d, M\right\} + \eta(1 - \delta\left\{x_t^d, M\right\})\delta\left\{j^d, M\right\}\end{aligned}$$

where on the final line we have used the fact that $p_{1|t}^{\theta}(x_1^d = x_t^d|x_t^{1:D}) = 0$ for $x_t^d = M$ and $p_{1|t}^{\theta}(x_1^d = x_t^d|x_t^{1:D}) = 1$ when $x_t^d \neq M$ because if a dimension is unmasked then it must be the true x_1 value under our definition of $p_{t|1}(x_t|x_1)$. We now find our transition probabilities

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \delta\left\{j^d, x_t^d\right\} + R_t^{\theta d}(x_t^{1:D}, j^d)\Delta t$$

For $j^d \neq x_t^d$,

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \Delta t\frac{1+\eta t}{1-t}p_{1|t}^{\theta}(x_1^d = j^d|x_t^{1:D})\delta\left\{x_t^d, M\right\} + \Delta t\eta(1 - \delta\left\{x_t^d, M\right\})\delta\left\{j^d, M\right\}$$

and for $j^d = x_t^d$

$$\begin{aligned}p_{t+\Delta t|t}(j^d = x_t^d|x_t^{1:D}) &= 1 - \sum_{k\neq x_t^d} p_{t+\Delta t|t}(k|x_t^{1:D}) \\ &= 1 - \sum_{k\neq x_t^d}\left(\Delta t\frac{1+\eta t}{1-t}p_{1|t}^{\theta}(x_1^d = k|x_t^{1:D})\delta\left\{x_t^d, M\right\} + \Delta t\eta(1 - \delta\left\{x_t^d, M\right\})\delta\left\{k, M\right\}\right)\end{aligned}$$

$$= 1 - \Delta t \frac{1+\eta t}{1-t} \delta\left\{x_t^d, M\right\} - \Delta t \eta (1 - \delta\left\{x_t^d, M\right\})$$

where again we have used the fact that $p^\theta_{1|t}(x_1^d = M | x_t^{1:D}) = 0$. Inspecting $p_{t+\Delta t|t}(j^d | x_t^{1:D})$ for $j^d \neq x_t^d$, we see that if $x_t^d = M$ then we have an overall probability of unmasking of $\frac{1+\eta t}{1-t}\Delta t$ and once we do unmask, the new value is drawn from $p^\theta_{1|t}(x_1^d | x_t^{1:D})$. This is like before but now there is a bonus probability of unmasking of $\frac{\eta t}{1-t}$. When $x_t^d \neq M$ then we have a probability of $\eta \Delta t$ of jumping back to the mask state. This creates a flux of states switching back and forth between masked and unmasked for $\eta > 0$ hence why these processes are more 'stochastic'. However, because when η is increased we also increase the rate at which we unmask, the desired conditional flow $p_{t|1}(x_t|x_1)$ is maintained for any value of η. Listing 3 shows PyTorch code that implements sampling with this extra stochasticity.

Listing 3. Masking sampling loop with noise

```
import torch
import torch.nn.functional as F
from torch.distributions.categorical import Categorical


# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S-1). We know
    the clean data contains no masks and hence we only need to output logits
    over the valid values.
t = 0.0
dt = 0.001
mask_index = S-1
N = 10 # Level of stochasticity

xt = mask_index * torch.ones((B, D), dtype=torch.long)

while t < 1.0:
    logits = model(xt, t * torch.ones((B,))) # (B, D, S-1)
    x1_probs = F.softmax(logits, dim=-1) # (B, D, S-1)
    x1 = Categorical(x1_probs).sample() # (B, D)
    will_unmask = torch.rand((B, D)) < (dt * (1 + N * t) / (1-t)) # (B, D)
    will_unmask = will_unmask * (xt == mask_index) # (B,D) only unmask currently
        masked positions
    will_mask = torch.rand((B, D)) < dt * N # (B, D)
    will_mask = will_mask * (xt != mask_index) # (B, D) only re-mask currently
        unmasked positions
    xt[will_unmask] = x1[will_unmask]
    t += dt
    if t < 1.0: # Don't re-mask on the final step
        xt[will_mask] = mask_index
```

Our method has similarities to other discrete diffusion models when using this form for $p_{t|1}$ and we clarify these links in App. H.2.

F.1.2. Purity Sampling

When using the masking form for $p_{t|1}$ we can also easily implement a purity sampling scheme (Tang et al., 2022). This sampling method decides which dimensions to unmask based on an estimate of the model confidence in that dimension's final value. Currently, our sampling method will uniformly at random choose which dimension to unmask. To improve upon this approach, purity sampling will instead rank dimensions based on which dimension has the highest model probability. More specifically, for each dimension we calculate a purity score for dimension d defined as

$$\text{purity}_d = \max_{x_1^d} \quad p^\theta_{1|t}(x_1^d | x_t^{1:D})$$

For the next simulation step, we then decide how many dimensions should be unmasked. The number of dimensions to unmask is binomially distributed with probability of success $\frac{\Delta t}{1-t}$ and number of trials equal to the number of dimensions

that are currently masked. Once we have sampled a number of dimensions to unmask from this binomial distribution, we then unmask that number of dimensions starting from the dimension with highest purity score, then the dimension with second highest purity score and so on. We only consider dimensions that are currently masked to be eligible for unmasking. When using $\eta > 0$, the probability of success in our binomial distribution increases to $\Delta t \frac{1+\eta t}{1-t}$ and so on average more dimensions get unmasked during each simulation step. At the end of each simulation step, we then remask a sample of randomly chosen dimensions which are uniformly chosen at random each with a probability $\Delta t \eta$ of being chosen.

F.2. Uniform Example

In this section we walk through the derivation and implementation of DFM when using the uniform based interpolation distribution. We start with the data conditional marginal distribution

$$\begin{aligned} p_{t|1}(x_t^{1:D}|x_1^{1:D}) &= \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d) \\ &= \prod_{d=1}^{D} \left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\frac{1}{S} \right) \end{aligned}$$

This distribution is all that is needed to train the denoising model $p_{1|t}^{\theta}(x_1^{1:D}|x_t^{1:D})$. We give PyTorch code for the training loop with the uniform interpolant in Listing 4.

Listing 4. Uniform training loop

```
import torch
import torch.nn.functional as F

# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S).

for x1 in dataset:
    # x1 has shape (B, D)
    optimizer.zero_grad()
    t = torch.rand((B,))
    xt = x1.clone()
    uniform_noise = torch.randint(0, S, (B, D))
    corrupt_mask = torch.rand((B, D)) < (1 - t[:, None])
    xt[corrupt_mask] = uniform_noise[corrupt_mask]
    logits = model(xt, t) # (B, D, S)
    loss = F.cross_entropy(logits.transpose(1,2), x1, reduction='mean')
    loss.backward()
    optimizer.step()
```

In order to sample our trained model, we will need to derive $R_t^{*d}(i^d, j^d|x_1^d)$. The first step is to find $\partial_t p_{t|1}(x_t^d|x_1^d)$,

$$\begin{aligned} \partial_t p_{t|1}(x_t^d|x_1^d) &= \partial_t \left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\frac{1}{S} \right) \\ &= \delta\left\{x_t^d, x_1^d\right\} - \frac{1}{S} \end{aligned}$$

We will now find $R_t^{*d}(x_t^d, j^d|x_1^d)$. As before we will always assume $x_t^d \neq j^d$ and calculate diagonal entries as needed using the relation $R_t(i,i) = -\sum_{j \neq i} R_t(i,j)$.

$$\begin{aligned} R_t^{*d}(x_t^d, j^d|x_1^d) &= \frac{\text{ReLU}\left(\partial_t p_{t|1}(j^d|x_1^d) - \partial_t p_{t|1}(x_t^d|x_1^d)\right)}{\mathcal{Z}_t^d p_{t|1}(x_t^d|x_1^d)} \\ &= \frac{\text{ReLU}\left(\delta\left\{j^d, x_1^d\right\} - \frac{1}{S} - \delta\left\{x_t^d, x_1^d\right\} + \frac{1}{S}\right)}{S\left(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\frac{1}{S}\right)} \end{aligned}$$

$$= \frac{\text{ReLU}\left(\delta\left\{j^d, x_1^d\right\} - \delta\left\{x_t^d, x_1^d\right\}\right)}{S(t\delta\left\{x_t^d, x_1^d\right\} + (1-t)\frac{1}{S}}$$

The only non-zero value is when $j^d = x_1^d$ and $x_t^d \neq x_1^d$ and so $R_t^{*d}(x_t^d, j^d | x_1^d)$ is

$$R_t^{*d}(x_t^d, j^d | x_1^d) = \frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\left(1 - \delta\left\{x_t^d, x_1^d\right\}\right)$$

We can now find the unconditional rate matrix, still assuming $x_t^d \neq j^d$

$$\begin{aligned} R_t^{\theta d}(x_t^{1:D}, j^d) &= \mathbb{E}_{p^\theta_{1|t}(x_1^d | x_t^{1:D})}\left[R_t^{*d}(x_t^d, j^d | x_1^d)\right] \\ &= \mathbb{E}_{p^\theta_{1|t}(x_1^d | x_t^{1:D})}\left[\frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}\left(1 - \delta\left\{x_t^d, x_1^d\right\}\right)\right] \\ &= \frac{1}{1-t}p^\theta_{1|t}(x_1^d = j^d | x_t^{1:D}) \end{aligned}$$

Our transition step is

$$p_{t+\Delta t|t}(j^d | x_t^{1:D}) = \delta\left\{j^d, x_t^d\right\} + R_t^{\theta d}(x_t^{1:D}, j^d)\Delta t$$

For $j^d \neq x_i^d$ this is

$$p_{t+\Delta t|t}(j^d | x_t^{1:D}) = \frac{\Delta t}{1-t}p^\theta_{1|t}(x_1^d = j^d | x_t^{1:D})$$

and for $j^d = x_t^d$ this is

$$\begin{aligned} p_{t+\Delta t|t}(j^d = x_t^d | x_t^{1:D}) &= 1 - \sum_{k \neq x_t^d} p_{t+\Delta t|t}(k | x_t^{1:D}) \\ &= 1 - \sum_{k \neq x_t^d} \frac{\Delta t}{1-t}p^\theta_{1|t}(x_1^d = k | x_t^{1:D}) \\ &= 1 - \frac{\Delta t}{1-t}\left(1 - p^\theta_{1|t}(x_1^d = x_t^d | x_t^{1:D})\right) \end{aligned}$$

Listing 5 shows PyTorch code that implements this sampling loop.

Listing 5. Uniform sampling loop

```
import torch
import torch.nn.functional as F
from torch.distributions.categorical import Categorical


# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S).
t = 0.0
dt = 0.001

xt = torch.randint(0, S, (B, D))
while t < 1.0:
    logits = model(xt, t * torch.ones((B,))) # (B, D, S)
    x1_probs = F.softmax(logits, dim=-1) # (B, D, S)

    # Calculate the off-diagonal step probabilities
    step_probs = ((dt / (1-t)) * x1_probs).clamp(max=1.0) # (B, D, S)

    # Calculate the on-diagnoal step probabilities
    # 1) Zero out the diagonal entries
    step_probs.scatter_(-1, xt[:, :, None], 0.0)
    # 2) Calculate the diagonal entries such that the probability row sums to 1
    step_probs.scatter_(-1, xt[:, :, None], (1.0 - step_probs.sum(dim=-1,
        keepdim=True)).clamp(min=0.0))

    xt = Categorical(step_probs).sample() # (B, D)

    t += dt
```

F.2.1. Detailed Balance

Here we derive the form of R_t^{DB} for the uniform interpolant case which we can use to vary the stochasticity of sampling. R_t^{DB} satisfies the detailed balance equation

$$p_{t|1}(i|x_1)R_t^{\mathrm{DB}}(i,j|x_1) = p_{t|1}(j|x_1)R_t^{\mathrm{DB}}(j,i|x_1)$$

We now make some assumptions for the form of R_t^{DB}. We will assume there will be some rate of transitions from x_1 back towards a random other state and a rate towards x_1 in order to cancel out this effect and achieve detailed balance. We note there are other choices for detailed balance, some of which we explore in App. H.1. We will again be assuming $i \neq j$ in the following calculations.

$$R_t^{\mathrm{DB}}(i,j|x_1) = a_t\delta\left\{i,x_1\right\} + b_t\delta\left\{j,x_1\right\}$$

We have parameterized R_t^{DB} with some time-dependent constants a_t and b_t. Substituting this into the detailed balance equation gives

$$\left(t\delta\left\{i,x_1\right\} + (1-t)\frac{1}{S}\right)\left(a_t\delta\left\{i,x_1\right\} + b_t\delta\left\{j,x_1\right\}\right) = \left(t\delta\left\{j,x_1\right\} + (1-t)\frac{1}{S}\right)\left(a_t\delta\left\{j,x_1\right\} + b_t\delta\left\{i,x_1\right\}\right)$$

Now, this equation must be true for any $i \neq j$. Pick $i = x_1$ and $j \neq x_1$ to get

$$\left(t + (1-t)\frac{1}{S}\right)a_t = (1-t)\frac{1}{S}b_t$$

$$b_t = a_t\frac{t + (1-t)\frac{1}{S}}{(1-t)\frac{1}{S}}$$

$$= a_t \frac{St + 1 - t}{1 - t} \tag{34}$$

We would obtain the same equation if we were to instead pick $i \neq x_1$ and $j = x_1$. Therefore we have to fix one of a_t or b_t. If we want a stochasticity level of η then we can set $a_t = \eta$ which is the rate at which points that are at the clean data come back off the clean datapoint. b_t can then be found from equation (34). This gives a form for R_t^{DB} of

$$R_t^{\mathrm{DB}}(i, j|x_1) = \eta\delta\left\{i, x_1\right\} + \eta\frac{St + 1 - t}{1 - t}\delta\left\{j, x_1\right\}$$

This can now be combined with R_t^{*d} to create a new unconditional rate matrix with a variable amount of stochasticity.

$$\begin{aligned}
R_t^{\theta d}(x_t^{1:D}, j^d) &= \mathbb{E}_{p^\theta_{1|t}(x_1^d|x_t^{1:D})}\left[R_t^{*d}(x_t^d, j^d|x_1^d) + {R^{\mathrm{DB}}}_t^d(x_t^d, j^d|x_1^d)\right] \\
&= \mathbb{E}_{p^\theta_{1|t}(x_1^d|x_t^{1:D})}\left[\frac{1}{1-t}\delta\left\{j^d, x_1^d\right\}(1 - \delta\left\{x_t^d, x_1^d\right\}) + \eta\delta\left\{x_t^d, x_1^d\right\} + \eta\frac{St + 1 - t}{1 - t}\delta\left\{j^d, x_1^d\right\}\right] \\
&= \mathbb{E}_{p^\theta_{1|t}(x_1^d|x_t^{1:D})}\left[\frac{1 + \eta + \eta(S-1)t}{1 - t}\delta\left\{j^d, x_1^d\right\}(1 - \delta\left\{x_t^d, x_1^d\right\}) + \eta\delta\left\{x_t^d, x_1^d\right\}\right] \\
&= \frac{1 + \eta + \eta(S-1)t}{1 - t}p^\theta_{1|t}(x_1^d = j^d|x_t^{1:D}) + \eta p^\theta_{1|t}(x_1^d = x_t^d|x_t^{1:D})
\end{aligned}$$

We can interpret this rate, with the first term being the rate at which we should transition to states that are predicted to correspond to the clean data. The second term is a 'noise term' which creates transitions away from the current state if it is predicted to correspond to the final clean data. The first term then has additional weighting as η is increased to counter act this effect. The effect of the stochasticity is then to create a flux going on and off the predicted final clean state during generation. We now find our transition probabilities

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \delta\left\{j^d, x_t^d\right\} + R_t^{\theta d}(x_t^{1:D}, j^d)\Delta t$$

For $j^d \neq x_t^d$,

$$p_{t+\Delta t|t}(j^d|x_t^{1:D}) = \Delta t\frac{1 + \eta + \eta(S-1)t}{1 - t}p^\theta_{1|t}(x_1^d = j^d|x_t^{1:D}) + \Delta t\eta p^\theta_{1|t}(x_1^d = x_t^d|x_t^{1:D})$$

We can find $p_{t+\Delta t|t}(j^d|x_t^{1:D})$ for $j^d = x_t^d$ programmatically as before by requiring that the probability vector sum to 1. Listing 6 shows the implementation for the uniform interpolant with noise.

Listing 6. Uniform sampling loop with noise

```
import torch
import torch.nn.functional as F
import torch.distributions.categorical import Categorical

# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S).

t = 0.0
dt = 0.001
noise = 1

xt = torch.randint(0, S, (B, D))

while t < 1.0:
    logits = model(xt, t * torch.ones((B,))) # (B, D, S)
    x1_probs = F.softmax(logits, dim=-1) # (B, D, S)
    x1_probs_at_xt = torch.gather(x1_probs, -1, xt[:, :, None]) # (B, D, 1)

    # Don't add noise on the final step
    if t + dt < 1.0:
        N = noise
    else:
        N = 0

    # Calculate the off-diagonal step probabilities
    step_probs = (
        dt * ((1 + N + N * (S - 1) * t ) / (1-t)) * x1_probs +
        dt * N * x1_probs_at_xt
    ).clamp(max=1.0) # (B, D, S)

    # Calculate the on-diagnoal step probabilities
    # 1) Zero out the diagonal entries
    step_probs.scatter_(-1, xt[:, :, None], 0.0)
    # 2) Calculate the diagonal entries such that the probability row sums to 1
    step_probs.scatter_(-1, xt[:, :, None], (1.0 - step_probs.sum(dim=-1,
        keepdim=True)).clamp(min=0.0))

    xt = Categorical(step_probs).sample() # (B, D)

    t += dt
```

F.3. General Case

We now describe the training and sampling loop for a general conditional flow $p_{t|1}(x_t|x_1)$. We require this interpolant to be factorized, $p_{t|1}(x_t^{1:D}|x_1^{1:D}) = \prod_{d=1}^{D} p_{t|1}(x_t^d|x_1^d)$, be differentiable and have $p_{t|1}(j^d|x_1^d) = 0 \implies \partial_t p_{t|1}(j^d|x_1^d) = 0$. We assume that we have access to functions that can sample from $p_{t|1}(x_t|x_1)$, evaluate $p_{t|1}(x_t|x_1)$ and evaluate $\partial_t p_{t|1}(x_t|x_1)$. Our training loop consists of sampling data, sampling $x_t \sim p_{t|1}(x_t|x_1)$ and training with the cross entropy loss, see Listing 7.

Listing 7. General training loop

```
import torch
import torch.nn.functional as F

# Variables, B, D, S for batch size, number of dimensions and state space size
    respectively
# Assume we have a model that takes as input xt of shape (B, D) and time of
    shape (B,) and outputs x1 prediction logits of shape (B, D, S).

def sample_p_xt_g_x1(x1, t):
    # x1 (B, D)
    # t (B,)
    # Returns xt (B, D)

for x1 in dataset:
    # x1 has shape (B, D)
    optimizer.zero_grad()
    t = torch.rand((B,))
    xt = sample_p_xt_g_x1(x1, t)
    logits = model(xt, t) # (B, D, S)
    loss = F.cross_entropy(logits.transpose(1,2), x1, reduction='mean')
    loss.backward()
    optimizer.step()
```

Now for sampling we can programmatically calculate $R_t^{*d}(x_t^d, j^d|x_1^d)$ using Eq. (27). It may not be possible to analytically calculate the expectation with respect to $p_{1|t}^\theta(x_1^{1:D}|x_t^{1:D})$ but we note that our Euler step is still valid if we instead take a sample from $p_{1|t}^\theta(x_1^{1:D}|x_t^{1:D})$ and substitute into $R_t^d(x_t^d, j^d|x_1^d)$, see App. G. We assume access further to a function that can produce samples from the prior distribution p_{noise} corresponding to the chosen $p_{t|1}$. We provide the general case sampling loop in Listing 8.

Listing 8. General sampling loop

```
def dt_p_xt_g_xt(x1, t):
    # x1 (B, D)
    # t float
    # returns (B, D, S) for varying x_t value

def p_xt_g_x1(x1, t):
    # x1 (B, D)
    # t float
    # returns (B, D, S) for varying x_t value

def sample_prior(num_samples, D):
    # num_samples, D both integer
    # returns prior sample of shape (num_samples, D)

t = 0.0
dt = 0.001
num_samples = 1000
xt = sample_prior(num_samples, D)

while t < 1.0:
    logits = model(xt, t * torch.ones((num_samples,))) # (B, D, S)
    x1_probs = F.softmax(logits, dim=-1) # (B, D, S)
    x1 = Categorical(x1_probs).sample() # (B, D)

    # Calculate R_t^*
    # For p(x_t | x_1) > 0 and p(j | x_1) > 0
    # R_t^*(x_t, j | x_1) = Relu( dtp(j | x_1) - dtp(x_t | x_1)) / (Z_t * p(x_t
       | x_1))
    # For p(x_t | x_1) = 0 or p(j | x_1) = 0 we have R_t^* = 0
    # We will ignore issues with diagnoal entries as later on we will set
    # diagnoal probabilities such that the row sums to one later on.

    dt_p_vals = dt_p_xt_g_xt(x1, t) # (B, D, S)
    dt_p_vals_at_xt = dt_p_vals.gather(-1, xt[:, :, None]).squeeze(-1) # (B, D)

    # Numerator of R_t^*
    R_t_numer = F.relu(dt_p_vals - dt_p_vals_at_xt[:, :, None]) # (B, D, S)

    pt_vals = p_xt_g_x1(x1, t) # (B, D, S)
    Z_t = torch.count_nonzero(pt_vals, dim=-1) # (B, D)
    pt_vals_at_xt = pt_vals.gather(-1, xt[:, :, None]).squeeze(-1) # (B, D)

    # Denominator of R_t^*
    R_t_denom = Z_t * pt_vals_at_xt # (B, D)

    R_t = R_t_numer / R_t_denom[:, :, None] # (B, D, S)

    # Set p(x_t | x_1) = 0 or p(j | x_1) = 0 cases to zero
    R_t[ (pt_vals_at_xt == 0.0)[:, :, None].repeat(1, 1, S)] = 0.0
    R_t[ pt_vals == 0.0] = 0.0

    # Calculate the off-diagonal step probabilities
    step_probs = (R_t * dt).clamp(max=1.0) # (B, D, S)

    # Calculate the on-diagnoal step probabilities
    # 1) Zero out the diagonal entries
    step_probs.scatter_(-1, xt[:, :, None], 0.0)
    # 2) Calculate the diagonal entries such that the probability row sums to 1
    step_probs.scatter_(-1, xt[:, :, None], (1.0 - step_probs.sum(dim=-1,
        keepdim=True)).clamp(min=0.0))

    xt = Categorical(step_probs).sample() # (B, D)
    t += dt
```

F.3.1. DETAILED BALANCE

There are many ways one could solve the detailed balance equation for R_t^{DB} as the choice will depend on what kinds of noise are desirable to include in the generative process. A baseline example of how you could solve the detailed balance equation for generate $p_{t|1}(x_t|x_1)$ is to note

$$\begin{aligned} R_t^{\text{DB}}(i,j|x_1)p_{t|1}(i|x_1) &= R_t^{\text{DB}}(j,i|x_1)p_{t|1}(j|x_1) \\ \frac{R_t^{\text{DB}}(i,j|x_1)}{R_t^{\text{DB}}(j,i|x_1)} &= \frac{p_{t|1}(i|x_1)}{p_{t|1}(j|x_1)} \end{aligned}$$

which gives a relation between the diagonal elements of R_t^{DB}. As a first choice we could simply set the upper triangular section of R_t^{DB} to 1 and set the lower triangular part to the ratio $\frac{p_{t|1}(i|x_1)}{p_{t|1}(j|x_1)}$ which would satisfy detailed balance.

G. CTMC Sampling Methods

In the main text, our sampling algorithm Alg. 1 first constructs the unconditional rate matrix $R_t^\theta(x_t, j) = \mathbb{E}_{p_{1|t}^\theta(x_1|x_t)}\left[R_t(x_t, j|x_1)\right]$ and then samples the next state from the Euler step,

$$x_{t+\Delta t} \sim \text{Cat}\left(\delta\left\{x_t, x_{t+\Delta t}\right\} + R_t^\theta(x_t, x_{t+\Delta t})\Delta t\right).$$

The form of this update means that we don't necessarily need to calculate the full expectation over $R_t(x_t, j|x_1)$. We can simply sample x_1 from $p_{1|t}^\theta(x_1|x_t)$ and then plug this sample into $R_t(x_t, j|x_1)$ which we then use in the Euler update. To see that this strategy still samples from the same distribution over $x_{t+\Delta t}$, we can write the distribution over $x_{t+\Delta t}$ as $p_{t+\Delta t|t}$,

$$\begin{aligned} p_{t+\Delta t|t}(x_{t+\Delta t}|x_t) &= \delta\left\{x_t, x_{t+\Delta t}\right\} + \mathbb{E}_{p_{1|t}^\theta(x_1|x_t)}\left[R_t(x_t, x_{t+\Delta t}|x_1)\right]\Delta t \\ &= \mathbb{E}_{p_{1|t}^\theta(x_1|x_t)}\left[\delta\left\{x_t, x_{t+\Delta t}\right\} + R_t(x_t, x_{t+\Delta t}|x_1)\Delta t\right] \\ &= \sum_{x_1} p_{1|t}^\theta(x_1|x_t)p_{t+\Delta t|t}(x_{t+\Delta t}|x_1, x_t) \end{aligned}$$

where

$$p_{t+\Delta t|t}(x_{t+\Delta t}|x_1, x_t) := \delta\left\{x_t, x_{t+\Delta t}\right\} + R_t(x_t, j|x_1)\Delta t$$

and so $p_{t+\Delta t|t}(x_{t+\Delta t}|x_t)$ can be seen as the marginal of joint distribution $p_{1|t}^\theta(x_1|x_t)p_{t+\Delta t|t}(x_{t+\Delta t}|x_1, x_t)$. Therefore, to produce a sample $x_{t+\Delta t}$ from $p_{t+\Delta t|t}(x_{t+\Delta t}|x_t)$, we can instead sample $x_1, x_{t+\Delta t}$ from the joint distribution $p_{1|t}^\theta(x_1|x_t^{1:D})p_{t+\Delta t|t}(x_{t+\Delta t}|x_1, x_t)$, and take only the $x_{t+\Delta t}$ part of this joint sample.

Another method to simulate a CTMC is τ-leaping, (Gillespie, 2001; Campbell et al., 2022) which allows multiple jumps to be made both across dimensions and within each dimension. Multiple jumps within a single dimension does not make sense for categorical data where there is no ordering, however, it can be useful for ordinal data such as a discretized image where the τ-leaping update allows multiple jumps to be applied at once to cover a larger distance. To calculate a τ-leaping update, a Poisson random variable needs to be drawn with the rate matrix giving the rate parameter. Therefore, for this type of update, the full unconditional $R_t^\theta(x_t, j)$ would need to be calculated.

We finally note that there is a body of work creating CTMC samplers for generative models (Sun et al., 2023b; Lou et al., 2023) that may be faster to simulate than the standard Euler step. In this work, we focus on framework simplicity, not optimizing for sampling speed and leave application of these approaches as future work.

H. Comparison with Discrete Diffusion Models

In this section we clarify the relationship between DFM and classical discrete diffusion models. In App. H.1 we compare to continuous time models using the uniform corruption process as an example. In App. H.2 we compare to discrete time models using the masking process as the example.

H.1. Continuous Time Discrete Diffusion Models

Here we compare to continuous time discrete diffusion models (Campbell et al., 2022) using the uniform corruption process as an example. In this section, we will assume $t = 0$ is pure noise and $t = 1$ is clean data which we note is a flipped definition of time to classical diffusion models to aid in our comparison with DFMs.

For discrete diffusion, we first specify a corruption process and then approximate its time reversal to give us the generative process. Our corruption process will evolve from $t = 1$ back to time $t = 0$. It will be specified using a rate matrix R_t. In order to make calculation of $p_{t|1}(x_t|x_1)$, R_t needs to be of a special form, namely $R_t = \beta(t)R_b$ where $\beta(t)$ is a time dependent scalar function and R_b is a base rate matrix that can be decomposed using the eigendecomposition $R_b = Q\Lambda Q^{-1}$. For uniform corruption, we can set $R_b = \mathbb{1}\mathbb{1}^\top - S\mathbb{I}$ where $\mathbb{1}$ is a vector of all 1's. We will now assume $S = 3$ so we can carry out all calculations explicitly.

We have $R_b = Q\Lambda Q^{-1}$ with

$$Q = \begin{bmatrix} -1 & -1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad \Lambda = \begin{bmatrix} -3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Q^{-1} = \begin{bmatrix} -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

To calculate $p_{t|1}(x_t|x_1)$ we can use the equation

$$P_t = Q \exp\left(\Lambda \int_1^t \beta(s)\mathrm{d}s\right) Q^{-1}$$

where $(P_t)_{ij} = p_{t|1}(x_t = j|x_1 = i)$ and exp is the element wise exponential. By the symmetry of the problem, we can infer that $p_{t|1}(x_t = j|x_1 = i)$ will have only two possible values. Either $j = i$ and we are finding the probability of staying at i, or $j \neq i$ and we are finding the probability of having left i, and since uniform corruption treats all states equally, these will be same quantities for any starting state and any state $j \neq i$. So to find our schedule we just need to consider one element of the matrix P_t. Let us consider an off-diagonal element $i \neq j$ of P_t, which will have probability

$$(P_t)_{ij} = \frac{1}{3}\left(1 - \exp\left(-3\int_1^t \beta(s)\mathrm{d}s\right)\right), \quad i \neq j$$

We will try and match this to the simple linear schedule that we have had as our running example in the explanation of DFM.

$$\begin{aligned} &\frac{1}{3}\left(1 - \exp\left(-3\int_t^1 \beta(s)\mathrm{d}s\right)\right) = \frac{1}{3}(1-t) \\ &\implies \beta(t) = \frac{1}{3t} \end{aligned}$$

Therefore, we have found that a corruption rate matrix of $R_t = \frac{1}{3t}\left(\mathbb{1}\mathbb{1}^\top - 3\mathbb{I}\right)$ gives a conditional flow of $p_{t|1}(x_t|x_1) = t\delta\left\{x_t, x_1\right\} + (1-t)\frac{1}{3}$.

The next step in a discrete diffusion model is to find the time reversed rate matrix $\hat{R}_t$ which gives a CTMC that runs in the opposite direction to R_t and generates novel data from noise. Here $\hat{R}_t$ is running from time $t = 0$ at noise towards clean data at $t = 1$. From Campbell et al. (2022), we have

$$\hat{R}_t(i,j) = \sum_{x_1} R_t(j,i)\frac{p_{t|1}(j|x_1)}{p_{t|1}(i|x_1)}p_{1|t}(x_1|i) \quad i \neq j$$

We notice a similarity to the DFM equations, where the generative rate is the expectation of a quantity with respect to $p_{1|t}(x_1|i)$. Indeed we now show that $R_t(j,i)\frac{p_{t|1}(j|x_1)}{p_{t|1}(i|x_1)}$ is a x_1 conditioned rate matrix $R_t^{\text{diff}}(i,j|x_1)$ that achieves the conditional flow $p_{t|1}(i|x_1)$. Consider the Kolmogorov equation

$$\partial_t p_{t|1}(i|x_1) = \sum_{j \neq i} R_t^{\text{diff}}(j,i|x_1)p_{t|1}(j|x_1) - \sum_{j \neq i} R_t^{\text{diff}}(i,j|x_1)p_{t|1}(i|x_1)$$

Substitute in our form for R_t^{diff}

$$\begin{aligned}
\text{RHS} &= \sum_{j\neq i} R_t(i,j)\frac{p_{t|1}(i|x_1)}{p_{t|1}(j|x_1)}p_{t|1}(j|x_1) - \sum_{j\neq i} R_t(j,i)\frac{p_{t|1}(j|x_1)}{p_{t|1}(i|x_1)}p_{t|1}(i|x_1) \\
&= \sum_{j\neq i} R_t(i,j)p_{t|1}(i|x_1) - \sum_{j\neq i} R_t(j,i)p_{t|1}(j|x_1) \\
&= -\left[\sum_{j\neq i} R_t(j,i)p_{t|1}(j|x_1) - \sum_{j\neq i} R_t(i,j)p_{t|1}(i|x_1)\right] \\
&= -\left[-\partial_t p_{t|1}(i|x_1)\right] \\
&= \text{LHS}
\end{aligned}$$

where on the second to last line we have used the fact that the corruption matrix $R_t(i,j)$ when started at $p_{t=1}(x_t|x_1) = \delta\{x_t, x_1\}$ will evolve the marginals according to $p_{t|1}(x_t|x_1)$ because this is how we derived $p_{t|1}(x_t|x_1)$ in the first place. Note R_t runs in the reverse direction hence the negative sign.

Therefore, the diffusion framework has made an implicit choice for $R_t(i,j|x_1) = R_t^{\text{diff}}(i,j|x_1)$ and this choice is made at training time. We now show on our uniform noise example that R_t^{diff} is simply $R_t^* + R_t^{\text{DB}}$ for a specific choice of R_t^{DB}.

Firstly, we write out the explicit form for R_t^{diff} using $R_t^{\text{diff}}(i,j|x_1) = R_t(j,i)\frac{p_{t|1}(j|x_1)}{p_{t|1}(i|x_1)}$, $R_t = \frac{1}{3t}\left(\mathbb{1}\mathbb{1}^\top - 3\mathbb{I}\right)$ and $p_{t|1}(i|x_1) = t\delta\{x_t, x_1\} + (1-t)\frac{1}{3}$.

$$R_t^{\text{diff}} = \frac{1}{3t}\begin{bmatrix} -1-\frac{1+2t}{1-t} & \frac{1+2t}{1-t} & 1 \\ \frac{1-t}{1+2t} & -2\frac{1-t}{1+2t} & \frac{1-t}{1+2t} \\ 1 & \frac{1+2t}{1-t} & -1-\frac{1+2t}{1-t} \end{bmatrix}$$

We will now find R_t^{DB} such that $R_t^{\text{diff}} = R_t^* + R_t^{\text{DB}}$. We will need a slightly more general form for R_t^{DB} than was previously derived for the uniform noise case. We will have

$$R_t^{\text{DB}}(i,j|x_1) = a_t\delta\{i,x_1\} + b_t\delta\{j,x_1\} + c_t(1-\delta\{i,x_1\})(1-\delta\{j,x_1\})$$

Using the detailed balance equation, $p_{t|1}(i|x_1)R_t^{\text{DB}}(i,j|x_1) = p_{t|1}(j|x_1)R_t^{\text{DB}}(j,i|x_1)$, we find that we need

$$a_t = \frac{(1-t)\frac{1}{3}b_t}{t + (1-t)\frac{1}{3}}$$

with b_t and c_t being fully flexible (provided they are positive). Using the form for $R_t^*(i,j|x_1) = \frac{1}{1-t}\delta\{j,x_1\}(1-\delta\{i,x_1\})$ that we derived in Appendix F.2 we have

$$R_t^* + R_t^{\text{DB}} = \begin{bmatrix} -\frac{1}{1-t} - b_t - c_t & \frac{1}{1-t} + b_t & c_t \\ \frac{(1-t)\frac{1}{3}b_t}{t+(1-t)\frac{1}{3}} & -2\frac{(1-t)\frac{1}{3}b_t}{t+(1-t)\frac{1}{3}} & \frac{(1-t)\frac{1}{3}b_t}{t+(1-t)\frac{1}{3}} \\ c_t & \frac{1}{1-t} + b_t & -c_t - \frac{1}{1-t} - b_t \end{bmatrix}$$

which is equal to R_t^{diff} if we have $b_t = c_t = \frac{1}{3t}$.

In summary, we have found that classical discrete diffusion models make an implicit choice for $R_t(i,j|x_1)$ which corresponds to a certain level of stochasticity in the CTMC and that the choice is made at training time because the rate matrix is used in the ELBO objective. Further, we have seen it is much harder to derive the noise schedule $p_{t|1}(x_t|x_1)$ in classical discrete diffusion models due to the need to be able to apply the matrix exponential to R_t. In DFM, we can simply write down the $p_{t|1}(x_t|x_1)$ noise schedule we want and we are not restricted in having to pick R_t that are amenable to matrix exponentiation. We also get to choose any $R_t(i,j|x_1)$ at test time rather than being fixed to the implicit choice of R_t^{diff}.

H.2. Discrete Time Discrete Diffusion Models

In this section we will clarify the link to the discrete time diffusion method D3PM (Austin et al., 2021) when using the masking process for both methods. Here, we will use the convention from Austin et al. (2021) of using $t = 0$ for clean data and $t = T$ for noise.

We will first summarize the key results from (Austin et al., 2021) when using the absorbing state process which is a different name for a masking type process (the mask is the absorbing state). t can take on any discrete value in $t \in \{0, 1, \dots, T\}$. The diffusion model is first defined using a noising transition kernel

$$p(x_t|x_{t-1}) = \begin{cases} 1 & \text{if } x_t = x_{t-1} = M \\ 1-\beta_t & \text{if } x_t = x_{t-1} \neq M \\ \beta_t & \text{if } x_t = M, x_{t-1} \neq M \end{cases}$$

From this transition kernel, we can then calculate the noise marginals, $p(x_t|x_0)$

$$p(x_t|x_0) = \left(1 - \prod_{k \leq t}(1-\beta_k)\right) \delta\{x_t, M\} + \left(\prod_{k \leq t}(1-\beta_k)\right) \delta\{x_t, x_0\}$$

We then define our generative reverse process as

$$p_\theta(x_{t-1}|x_t) = \sum_{x_0} p(x_{t-1}|x_t, x_0) p_\theta(x_0|x_t)$$

where $p_\theta(x_0|x_t)$ is the learned denoising model. Note how this is similar to our generative process, $R_t^\theta(x_t, j) = \mathbb{E}_{p_\theta(x_1|x_t)}[R_t(x_t, j|x_1)]$ where now $p(x_{t-1}|x_t, x_0)$ is the transition kernel for the clean data conditioned process. We then create our generative model by taking the expectation of this conditional kernel with respect to our denoising model.

Continuing with the D3PM example using the absorbing state process, we obtain the following form for $p_\theta(x_{t-1}|x_t)$

$$p_\theta(x_{t-1}|x_t) = \begin{cases} \frac{1-\prod_{k \leq t-1}(1-\beta_k)}{1-\prod_{k \leq t}(1-\beta_k)} & \text{if } x_t = x_{t-1} = M \\ \frac{\beta_t \prod_{k \leq t-1}(1-\beta_k)}{1-\prod_{k \leq t}(1-\beta_k)} p_\theta(x_0 = x_{t-1}|x_t) & \text{if } x_t = M, x_{t-1} \neq M \\ \delta\{x_{t-1}, x_t\} & \text{if } x_t \neq M \end{cases}$$

When we set $\beta_t = \frac{1}{T-t+1}$, we obtain a linear noise schedule giving

$$p_\theta(x_{t-1}|x_t) = \begin{cases} \left(1 - \frac{1}{t}\right) & \text{if } x_t = x_{t-1} = M \\ \frac{1}{t} p_\theta(x_0 = x_{t-1}|x_t) & \text{if } x_t = M, x_{t-1} \neq M \\ \delta\{x_{t-1}, x_t\} & \text{if } x_t \neq M \end{cases}$$

Now, let us define $\xi := \frac{t}{T}$ to be the proportion that the process is through the total number of time steps. $\xi \in [0, 1]$ and if we consider it to be an analogue of our continuous time variable, we can see that the original discretization steps of D3PM correspond to a discretization of the $[0, 1]$ interval with timesteps of $\Delta t = \frac{1}{T}$. Substituting these definitions into our update step gives,

$$p_\theta(x_{t-1}|x_t) = \begin{cases} \left(1 - \frac{\Delta t}{\xi}\right) & \text{if } x_t = x_{t-1} = M \\ \frac{1}{\xi} \Delta t p_\theta(x_0 = x_{t-1}|x_t) & \text{if } x_t = M, x_{t-1} \neq M \\ \delta\{x_{t-1}, x_t\} & \text{if } x_t \neq M \end{cases}$$

Now we can see a clear comparison to Eq. (33) noting the flipped definition of time. With our method we can pick any time discretization at test time because our method has been trained on all possible $t \in [0, 1]$. We also derive R_t^{DB} for the masking case which is not included in the prior D3PM framework. For training we note that the ELBO also simplifies down to a weighted cross entropy term for D3PM as noted by (Austin et al., 2021) and is also the case in our framework, see Appendix C.2.1.

I. Text Experiment Details

Code for our text experiments can be found at `https://github.com/andrew-cr/discrete_flow_models`.

For our denoising network we use the transformer architecture (Vaswani et al., 2017) as implemented in the nanoGPT repository, `https://github.com/karpathy/nanoGPT`. We generally follow the smallest GPT2 architecture

(Radford et al., 2019). At the input we have our input tokens x_t of shape B, D where B is the batch size and D is the number of dimensions i.e. the sequence length, our time t of shape B, and, if we are self-conditioning, prior x_1 prediction tokens of shape B, D. We embed the x_t and x_1 tokens using the same learned embedding, and use a model embedding size of 768 resulting in tensors of shape $B, D, 768$. We embed the position of each token using a learned embedding for each possible position. We embed the time t, using Transformer sinusoidal embeddings following (Ho et al., 2020). We train all our diffusion models with self-conditioning (Chen et al., 2023a). To input the prior x_1 prediction, we stack the x_t embedded tensor $B, D, 768$ with the x_1 prior prediction token tensor $B, D, 768$ to obtain a tensor of shape $B, D, 768 \times 2$. We then apply a linear layer to project down to the model embedding dimension resulting in a tensor of shape $B, D, 768$. Before applying transformer blocks, we add together the x_t (and x_1) embedding tensor, the position embedding and the time embedding to obtain the final $B, D, 768$ input tensor.

The transformer stack consists of 12 transformer blocks, each block consisting of a LayerNorm, SelfAttention, LayerNorm, MLP stack. Within our SelfAttention block, we use 12 heads and apply Qk-layernorm (Dehghani et al., 2023) to our query and key values as we observed this improved convergence. Our MLP blocks consist of a $768 \to 768 \times 4$ linear layer, followed by a GELU activation, followed by a $768 \times 4 \to 768$ linear layer. We do not apply dropout. Our output layer consists of a linear head with output dimension 28. We use 28 token categories, 26 lower case letters, a whitespace character and a mask token. The model outputs logits of shape $B, D, 28$ which we then apply a softmax to, to obtain $p_\theta(x_1|x_t)$ probabilities.

The dataset text8 is 100MB of text data from English Wikipedia. The text is all converted to lower case letters, i.e. capital letters are converted to lower case and numbers are written as text, i.e. 8 becomes 'eight'.

During training, we use a batch size of 256 with 8 gradient accumulation steps. We train on sequences of length 256. The model is therefore trained on $524,288$ tokens per gradient update. To train self-conditioning, on 50% of training iterations, we input prior x_1 prediction tokens as all masks so that the model learns to be able to predict x_1 without any prior information. On the other 50% of training iterations, we perform two model forward passes. We first predict x_1 using masks as the prior x_1 tokens to obtain an initial set of $p_\theta(x_1|x_t)$ logits. We then sample from the initial $p_\theta(x_1|x_t)$ distribution to obtain predicted x_1 tokens. We then feed these tokens back into the model through the self-conditioning input and predict the x_1 logits once more. These logits are then used in the loss. We only back propagate through the second forward pass of the model.

When training the D3PM model, we found that the default cross entropy weighting of $1/t$ (with a flipped definition of time) resulted in poor convergence and so we applied an equal weighting of the cross entropy across time to be consistent with the DFM loss.

We train our D3PM and DFM models for 750k iterations on 4 Nvidia A40 GPUs using a learning rate of 10^{-4} and 1000 linear warm up steps. We use a cosine decay schedule after the initial warm up towards a minimum learning rate of 10^{-5} which would be reached at 1M iterations. We use the AdamW optimizer (Loshchilov & Hutter, 2017) with weight decay parameter 0.1. We monitor the validation loss throughout training. Validation loss continues to drop throughout training and we evaluate the final $750k$ model in our experiments. When training the autoregressive model, we use the same architecture but find that it begins to overfit the data much faster than the diffusion based models. After 3500 iterations the validation loss begins to increase and so we use the model with minimum validation loss in our evaluations. This is consistent with findings that autoregressive models require much less compute to converge than diffusion based models (Gulrajani & Hashimoto, 2023).

We use the masking interpolant in our DFM with linear interpolant, as described in Appendix F.1. For D3PM, we use the absorbing state corruption process, the links to the DFM process are described in Appendix H.2.

For the SEDD baseline, we train two models from scratch using the provided code for $750k$ training iterations with an effective batch size of 2048 to be consistent with the DFM and D3PM training runs. All other parameters are left at their default values with the transformer using the same hidden size, number of blocks and number of layers as our other runs.

For evaluation, we sample the DFM with $\Delta t = 0.001$. We simulate up to $t = 0.98$ and then for any remaining tokens that are still mask, we set them to the most likely token under the model's denoising distribution, $p_\theta(x_1|x_t)$. We stop simulating at $t = 0.98$ to avoid any singularities similar to how diffusion models stop near $t = 0$. For D3PM we train with 1000 timesteps to match DFM.

For each temperature setting applied to the $p_\theta(x_1|x_t)$ logits, we sample 512 sequences all of length 256 tokens. We then

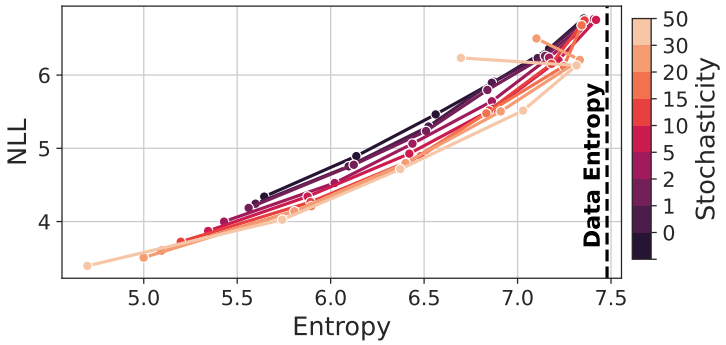


Figure 4. Curves in Entropy-NLL space for varying noise levels used during sampling. For each noise level, the temperature applied to the logits of the $p_\theta(x_1|x_t)$ prediction is varied over values $0.5, 0.6, 0.7, 0.8, 0.9, 1.0$.

Method	BPC
DFM $\eta = 0$	≤ 1.41
Multinomial Diffusion (Hoogeboom et al., 2021)	≤ 1.72
MAC (Shih et al., 2022)	≤ 1.40
BFN (Graves et al., 2023)	≤ 1.41
D3PM Uniform (Austin et al., 2021)	≤ 1.61
D3PM Absorb (Austin et al., 2021)	≤ 1.45
SEDD Uniform (Lou et al., 2023)	≤ 1.41
SEDD Absorb (Lou et al., 2023)	≤ 1.32

Table 5. Model log-likelihoods computed on the test set of text8 in bits-per-character (BPC).

calculate the negative log-likelihood assigned to each sequence using GPT-J-6B (Wang & Komatsuzaki, 2021) and the BPE tokenizer (Radford et al., 2019). We then average the negative log-likelihoods over the 512 sequences. The sample entropy is calculated by first tokenizing with the BPE tokenizer and then calculating the entropy as $\sum_i -p_i \log p_i$ where p_i is the empirical probability of token i estimated using the full set of 512 samples. Tokens for which $p_i = 0$ are not included in the sum. For reference, the dataset achieves a negative log-likelihood of 4.2 as measured by GPT-J-6B.

I.1. Stochasticity Sweep

Here we examine the effect of the noise level η on the sample quality of generations from our DFM method. We follow the follow the same procedure as before but vary η with values $\eta = 0, 1, 2, 5, 10, 15, 20, 30, 50$. We plot the results in Figure 4. We find that generally, as the noise level increases, we lower our negative log-likelihood. However, we find that if the noise level is increased too much, then degenerate behaviour can occur, for example when $\eta = 50$, at high logit temperatures the negative log-likelihood increases and the sample entropy decreases away from the dataset. Observing the samples, we find that the model generates incoherent text at this point. We find that the intermediate noise level $\eta = 15$ provides good sample quality whilst avoiding this behaviour.

I.2. Model Log-Likelihoods

Here we calculate bounds on the log likelihood $\log p_\theta(x_1)$ that the model assigns to the test set of text8. We use Eq. (24) to calculate this bound. We compare our log-likelihoods to other discrete diffusion style methods in terms of bits-per-character in Table. 5, reporting the numbers from Lou et al. (2023). We find that DFM achieves a similar BPC to previous masking style diffusion models with the recent work of Lou et al. (2023) achieving the lowest BPC.

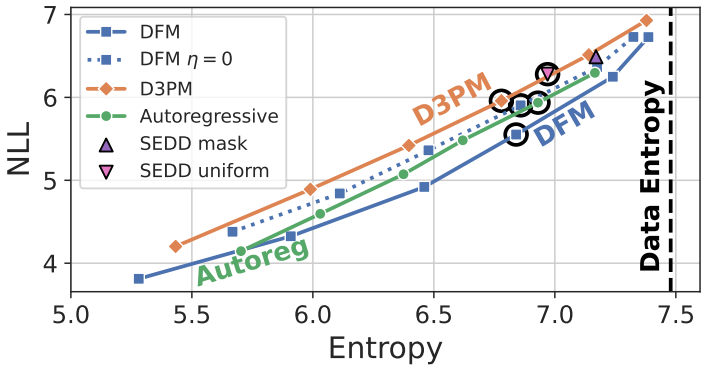


Figure 5. The temperature settings for each model for which we will examine sample generations. The selected temperature setting is highlighted with a black circle.

I.3. Example Text Generations

In this section we provide non cherry picked generations from the text models. For each model we have swept over the temperature applied to the logits and it would be impractical to include examples for all models for all temperature settings. Instead, we select one temperature setting for each model such that the samples have similar entropy but vary in negative log-likelihood. We show the selected temperature settings in Figure 5. For SEDD the method does not have a temperature setting and we select the corruption style that is closest in entropy-NLL space to the other methods.

SEDD Uniform
Samples:

change status regional courses and markets especially canada sport in canada and tennessee in canada the official light offered an newspaper licence named liu beijing world s main neighbouring fan site was sugar the only man with major historical works lic

ions of extension one or at least four subsets of a unique value of one example all these extensions heard of the function is called real line the implementation comes distributed with a continuous input extension to input and two classes the diagram emplo

of physics the radio atomic institutions independently the eastern united states followed into four six countries norway thus was the father of the university of gloucester but while also the father of germany can we also announce the coexistence of limit

D3PM Temperature 0.8
Samples:

ved as a personal area to form the five counties of the area and a country with their own which is usually called paris gietgothic can also lead an area to work in divisions over a pileur as in the name of man the bears have over the last two years from th

one five zero zero zero zero press money to present this to a meschasel linear industrial base ulse sudan expanded its economy and accounts for car prices and two eight five more than one zero zero of the largest industrial inventions over the world were

eed alternatively as human being and the anti constitutionalay doctrines a particular example of the concept is one reason for human rights or as in certain regions there is a double constitution more recognized region of europe in this region the glass an

DFM $\eta = 0$ Temperature 0.8
Samples:

ed era vol seven one nine one one december one nine six one junju that s one of nine one one country page of love footnote pages charles s feadman history of the red sea corea one nine nine one red sea vol one january one nine nine seven flying profiles ch

allowes the vectores to be composed as systems of data for example no machine is a computer one would do not know where there are undirected storage of other data storage particularly the computer science eve to substitute such a based data that is one of

me io the plate n and feminine along the trail to change the amount of naturated information in the start tape selective figurative memory the mind is determined by the second net on the string c with two buttons the tag retes the header when queued the se

Autoregressive Temperature 0.9
Samples:

licklyn american football coach to holy roman emperor and roman stories radio and facilities in the u s civil rights movement the dc circuit collection of the witches leading the transissario times and spinoffs to american cartoonists cartoonist kyle marci

the british one one eight four minamoto minister or al di nortello ministries son of monte oise klepe which chose to give up its character on the go he was known to publish a wade of white performances started in one eight five one kleine married the gigan

mausoleum in one eight one six alabama was engaged by a large scale as we know alabama migration the palace of westminsters and proceeded to father she also learned to speak with the abramic mouth of the space the replica was apparently built de provence g

DFM $\eta = 15$ Temperature 0.8
Samples:

e curous greek by alexander van hep ven see archaic origin of the word cupola another meaning suggests that the word kupola is the latin word cupei kupolum old german derived from the latin word for the river the name comes from a latin word for tree with

es so balloonists refine this combination specifically to preserve your own land in the runner both examples of clean steering creating agout like rods that produced successful rods and for the end the first few pistols compact stunt a musical setting mult

by reign over agassi is considered a greatest match by the day he will never play and will continue to be imitated agassi can play determinedly but agassi would always look to the victorious build he should not finish years going up to then that he would b

J. Protein Generation Experiment Details

We present additional experiment details and results for protein generation with Multiflow.

Code for Multiflow and experiments can be found at `https://github.com/jasonkyuyim/multiflow`

J.1. Experimental Details

Model Architecture. We use an architecture modified from the FrameDiff architecture from Yim et al. (2023b). This architecture consists of Invariant Point Attention (Jumper et al., 2021) combined with transformer blocks, we refer to Yim et al. (2023b) for in-depth details. We modify this network architecture by increasing the number of network blocks to 8, increasing the number of transformer layers within each block to 4, decreasing the number of hidden channels used in the IPA calculation to 16, removing skip connections and removing psi-angle prediction. To enable our model to output logits for the discrete $p^{\theta}_{1|t}(x_1|x_t)$ distribution, we add an output 3 layer MLP with the same embedding size as the main trunk. This results in a network with 21.8M parameters.

In Yim et al. (2023b), psi-angle prediction is used to infer the location of oxygen atoms, however, this position can be inferred to high accuracy using prior knowledge of the backbone structure of proteins, following (Yim et al., 2023a).

When training with our $t, \tilde{t}$ objective that enables the model to learn over different relative levels of corruption between structure and sequence, 10% of the time we set $t = 1$ and draw $\tilde{t} \sim \mathcal{U}(0, 1)$ and 10% of the time we set $\tilde{t} = 1$ and draw $t \sim \mathcal{U}(0, 1)$. The remaining 80% of the time we draw both t and $\tilde{t}$ independently from $t, \tilde{t} \sim \mathcal{U}(0, 1)$.

J.2. Additional Multiflow Results

We show results of Multiflow across more lengths than done in Sec. 6.2.1 and show that using the ESMFold oracle for data distillation still gives improved performance when we switch the evaluation oracle to AlphaFold2.

Main metrics with standard error. Table. 6 presents results of Table. 3 with standard error. We see our interpretation of the results do not change.

Larger length range. Our results in Sec. 6.2.1 only evaluated 4 lengths (70, 100, 200, 300) to match the benchmark in RFdiffusion. However, other works have evaluated designability across all the lengths the method was trained on. We follow Protpardelle (Chu et al., 2023) to use Multiflow in generating 8 samples per length in the range $\{50, 51, \ldots, 400\}$. Fig. 6 shows the results in the same format as Figure 2B in Protpardelle. We see Multiflow achieves near perfect designability up to around length 350 at which point designability starts to drop. This is expected since Multiflow was only trained on lengths up to 384, but also demonstrates the ability to generalize beyond the lengths it was trained on. We see Multiflow also achieves a desirable spread of secondary structure. We show samples above length 370 with the highest and lowest Co-design 1 RMSD in Fig. 7.

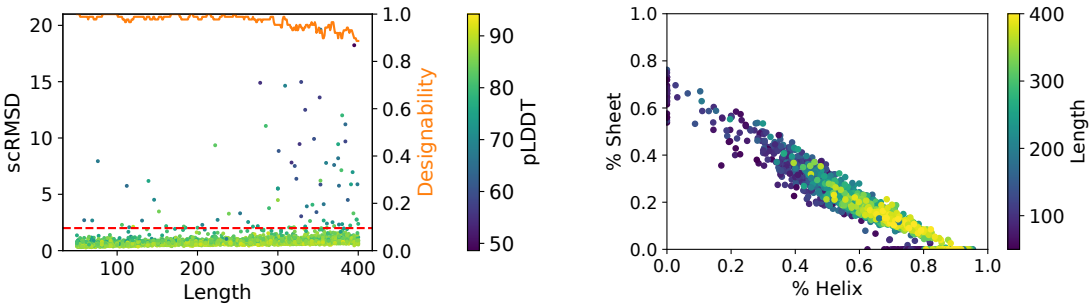


Figure 6. **Multiflow results on Protpardelle benchmark.** (**Left**) PMPNN 8 scRMSD and designability versus length. Designability is computed as the proportion of samples that have scRMSD < 2Å within a sliding window of size 11. Average pLDDT as computed by ESMFold for each sample is plotted as the colour of the scatter point. (**Right**) Secondary structure distribution. For each sample the proportion of residues as part of an alpha helix or beta strand is measured giving an xy scatter point coordinate.

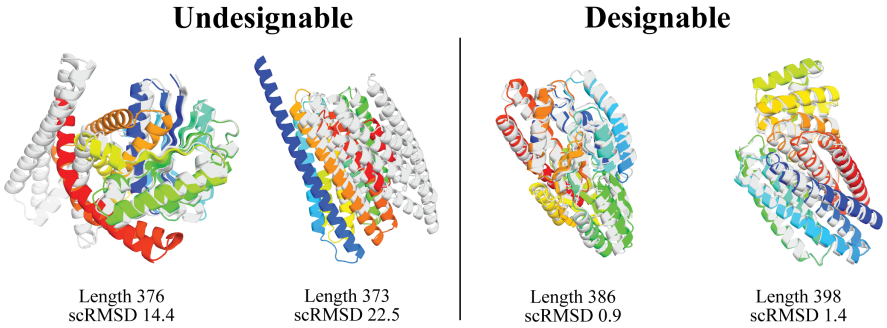


Figure 7. **Multiflow samples.** (**Left**) 2 undesignable Multiflow samples with the highest scRMSD from the benchmark. (**Right**) 2 designable Multiflow samples with the lowest scRMSD from the benchmark.

AlphaFold2 evaluation oracle. In Sec. 6.2.1, we presented a distillation technique of filtering out training examples that did

not pass the designability criterion. This also involved adding more proteins to the training set after sampling structures with Multiflow and filtering with designability using ProteinMPNN and ESMFold. A potential risk of distillation is our model may overfit to ESMFold since this model is used to filter training data and also for evaluation. We show this is not the case in Table. 7 by presenting the Co-design 1 results using AlphaFold2 (AF2) as an alternative oracle. Our main results do not use AF2 since it is very slow and cumbersome to run and evaluate all our baselines. We evaluated Multiflow with and without distillation to test *if distillation with ESMFold provides an improvement regardless of the oracle used at evaluation*. Overall designability numbers are lower with AF2; however, in both columns we see there is a two fold improvement regardless of the evaluation oracle. This demonstrates distillation is not overfitting to the oracle used at evalution.

Table 7. Co-design 1 designability results based on oracle.

	Designability with ESMFold	Designability with AF2
Multiflow w/o distillation	0.41	0.38
Multiflow w/ distillation	0.88	0.83
Net improvement	**+0.47**	**+0.45**

J.3. Uniform Conditional Flow Ablation

We ablate our use of the masking conditional flow and train a version of our Multiflow model using the uniform conditional flow (see App. F.2). We assessed the model's co-design performance by measuring the Co-Design 1 designability and diversity versus stochasticity level used at inference time. We also measure the secondary structure composition of the generated samples versus stochasticity level. Our results are given in Fig. 8. We find that in general, the Co-Design 1 designability increases with increasing stochasticity whilst the diversity as measured by the number of structural clusters decreases. We can see the reason when examining the secondary structure statistics versus stochasticity. We see that at high stochasticity levels, the model heavily favours generating alpha helices at the expense of beta strands thus reducing the overall structural diversity. This will be due to interactions between errors in the model and the 'churn' induced by extra stochasticity. It may be counter-intuitive that extra stochasticity reduces model diversity however we hypothesize that this is linked to the stochasticity inducing the model to converge on local optima in the likelihood landscape. When the model is generating a sample that it is confidence about, extra stochasticity will not shift it away from continuing down this simulation trajectory. However, when the model is exploring lower likelihood regions, the stochasticity can shift the models trajectory until it becomes stuck in a local optima again.

We find an overall worse trade-off between diversity and designability when using the uniform interpolant and so opt to use the masking interpolant in our main models.

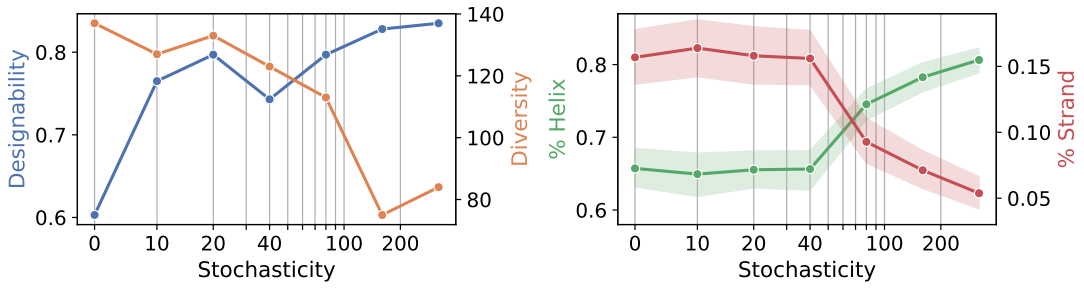


Figure 8. Sample metrics for Multiflow trained with the uniform interpolant on the discrete sequence modality. **(Left)** Co-Design 1 designability and diversity versus stochasticity level used when simulating the discrete CTMC. Higher is better for both designability and diversity. **(Right)** Average proportion of residues that are part of an alpha helix or beta strand versus the stochasticity level used to simulate the CTMC. Each point corresponds to the mean over 400 samples, 100 samples each for lengths 70, 100, 200, 300. Error bars show the standard error of the mean.

J.4. Forward and Inverse Folding Experiments

The goal of our work is to develop the missing piece for a general-purpose framework for protein generation – namely DFM to integrate discrete data generation with a flow model. We combined DFM and FrameFlow to develop Multiflow where we have flexibility at inference time to choose which modality to provide and which to generate. The task we focus on in this work is co-generation where the structure and sequence are jointly sampled rather than one after the other as done in prior works. The other useful tasks in protein modeling are forward and inverse folding. The two tasks are briefly described as follows; more in-depth description can be found in Gao et al. (2020).

1. **Forward folding**: the task is to take the sequence as input and predicts the most thermodynamically plausible structure of the sequence. During evaluation, the ground truth structure is known, so we calculate the aligned structure erorr between the prediction and the ground truth. Several metrics exist to compute structure error, such as the Global Distance Test (GDT) commonly used in biophysical modeling (Pereira et al., 2021). We choose to use the aligned backbone RMSD error to keep our analysis simple and intuitive. The most well-used methods are AlphaFold2 (Jumper et al., 2021), RosettaFold (Baek et al., 2021), and ESMFold (Lin et al., 2023). AlphaFold2 and RosettaFold rely on using evolutionary information which our model does not have access to (though can be extended to use). We compare against ESMFold, which does not use explicit evolutionary information, and due to its speed.
2. **Inverse folding**: the task is to use the structure as input and predict the most likely sequence that would *forward fold* into the structure. By this definition, the most sensible metric is the designability metric also used for co-generation. Specifically, the inverse folding model generates a sequence and we use ESMFold to predict the structure given this generated sequence. We call the self-consistency RMSD (scRMSD) as the RMSD between the structure predicted by ESMFold and the original input structure (Trippe et al., 2022). The objective is to minimize scRMSD. The de facto method for inverse folding is ProteinMPNN (Dauparas et al., 2022). Hence we compare against ProteinMPNN.

It is important to emphasize that different deep learning models have been *specifically* developed for forward and inverse folding, but no method can accomplish both tasks nor co-generate both sequence and structure. Multiflow is unique in this regard to be able to perform co-generation, forward folding, and inverse folding. We leave improving forward and inverse folding performance as a future work. **Our aim is to demonstrate baseline performance of using a co-generation method to perform forward and inverse folding.** We hope others can aid in advancing general purpose protein generative models.

Test set. ESMFold and ProteinMPNN have their own training and test sets which makes rigorous comparison impossible. Re-training ESMFold and ProteinMPNN with the same training set of Multiflow is beyond the scope of our work. Our results are a initial baseline of how Multiflow generally fares to specialized models on forward and inverse folding.

Our test set is based on a time-based split of the PDB. We downloaded structures and sequences from the PDB that were released between 1st September 2021 and 28th December 2023. *This time-based split ensures that none of the test set proteins are present in the training data for Multiflow, ProteinMPNN or ESMFold.* We then select all single chain monomeric proteins with length between 50 and 400 inclusive. We further filter out proteins that are more than 50% coil residues and proteins that have a radius of gyration in the 96th percentile of the original dataset or above. We also filter out structures that have missing residues. We cluster proteins using the 30% sequence identity MMSeqs2 clustering provided by RCSB.org. We take a single protein from each cluster that matches our filtering criteria. This gives us a test set of 449 proteins with minimum length 51 and maximum length 398.

J.4.1. Forward Folding Results

As described in Table. 2, forward folding with Multiflow is performed by fixing the sequence time to $\tilde{t} = 1$, providing the ground truth sequence as input, and running DFM from $t = 0$ to $t = 1$.

In Fig. 9 we examine the distribution of errors on our test set for both ESMFold and Multiflow. We find that generally Multiflow can have some success with proteins of smaller length but struggles with longer proteins. We investigate salient test examples from the plot to understand success and failure modes of our model. Multiflow is generally able to predict realistic protein structures with often similar secondary structure distributions as to the ground truth example seen by having similar proportions of non-loop residues between the ground truth and predicted structure. However, Multiflow often fails to predict the exact folded structure with high accuracy.

We quantify the secondary structure prediction accuracy in Fig. 10 by comparing the secondary structure present in the

ground truth versus the structure predicted by Multiflow. We find good correlation between the predicted secondary structure and ground truth highlighting that Multiflow is able to use information present within the given sequence to generate structures.

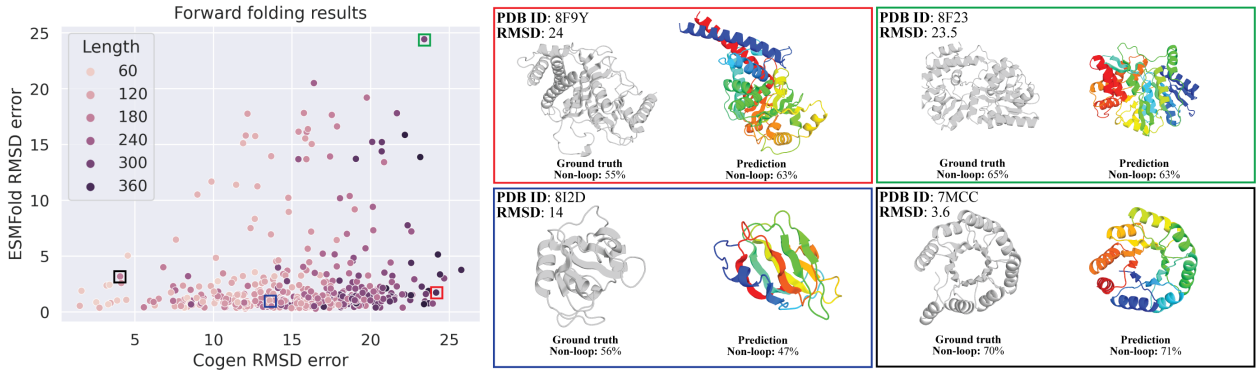


Figure 9. Forward folding RMSD metrics **(Left)** RMSD error between ground truth and predicted structures for Multiflow along the x-axis versus RMSD error for ESMFold on the y-axis. Each dot represents a protein in the test set. The shading of each point represents the length of the protein. **(Right)** Visualizations of ground truth structure (left) in grey and predicted structure (right) in color for 4 salient examples highlighted on the RMSD error plot. For each, the Multiflow RMSD error is given along with the proportion of non-loop residues for both the ground truth and prediction.

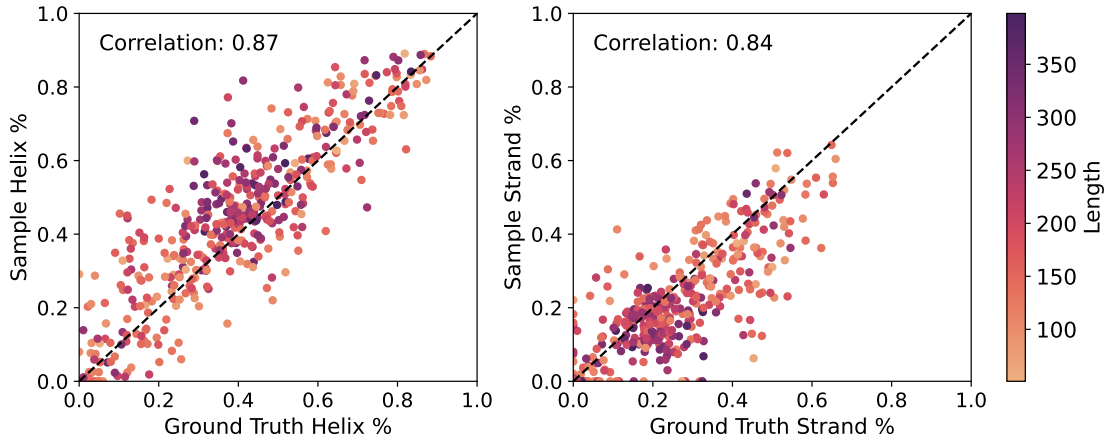


Figure 10. Proportion of residues that are part of secondary structure elements for both the Multiflow predicted structure and the ground truther structure. We plot the ground truth proportion of residues in a secondary structure element along the x-axis and the proportion of residues in the predicted structure on the y-axis. The left plot examines alpha helices whilst the right plot examines strand elements. Each scatter point represents a test set protein with the colour indicating the length. The perfect result of exactly matching proportion with the ground truth is plotted as a dashed diagonal line. We also report the correlation coefficient for each plot.

J.4.2. Inverse Folding Results

Similarly to forward folding, inverse folding with Multiflow is performed by fixing the structure time to $t = 1$, providing the ground truth structure and running the sequence flow from $\tilde{t} = 0$ to $\tilde{t} = 1$.

We plot our results in Fig. 11. We find that Multiflow performs competitively with PMPNN across a wide range of protein lengths with PMPNN achieving slightly lower scRMSD values on average. For both models, scRMSD tends to cluster around 1 to 2 scRMSD. There are test proteins for which PMPNN achieves a lower scRMSD and also cases protein for which Multiflow acheives the lower scRMSD.

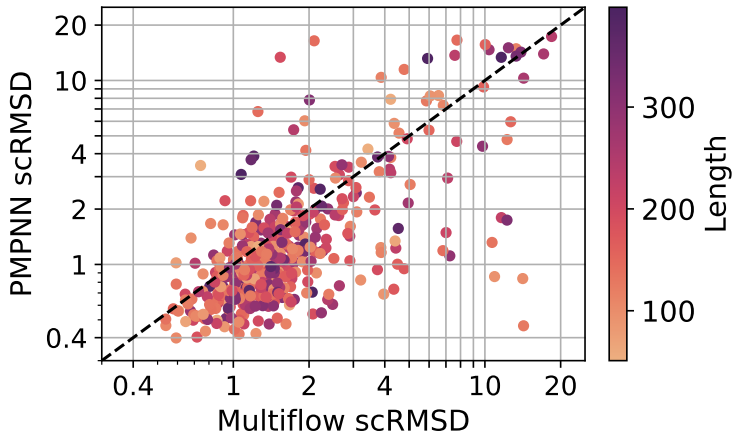


Figure 11. Multiflow scRMSD versus PMPNN scRMSD on our test set. Each scatter point represents a protein with the shading giving the length. We also plot the dividing line of equal scRMSD for the two models for ease of comparison.

Method	Co-design 1			PMPNN 8			PMPNN 1		
	Des.	**Div.**	**Nov.**	**Des.**	**Div.**	**Nov.**	**Des.**	**Div.**	**Nov.**
Protpardelle	0.04 (0.01)	5 (0.3)	0.69 (0.01)	0.9 (0.01)	47 (0.7)	0.59 (0.01)	0.63 (0.01)	38 (2.7)	0.60 (0.01)
ProteinGenerator	0.37 (0.01)	35 (2.6)	0.69 (0.01)	0.89 (0.00)	75 (1.5)	0.65 (0.02)	0.78 (0.02)	64 (4.6)	0.66 (0.01)
RFdiffusion				0.87 (0.02)	158 (3.7)	0.63 (0.01)	0.66 (0.02)	111 (5.9)	0.64 (0.01)
Cogen	0.86 (0.01)	143 (6.4)	0.61 (0.02)	0.99 (0.00)	156 (7.7)	0.61 (0.01)	0.88 (0.01)	143 (6.4)	0.61 (0.02)
Cogen w/o distillation	0.42 (0.02)	72 (2.4)	0.62 (0.01)	0.89 (0.00)	126 (1.0)	0.62 (0.02)	0.71 (0.02)	101 (7.1)	0.63 (0.02)
Cogen w/o sequence				0.99 (0.00)	116 (1.0)	0.63 (0.01)	0.86 (0.01)	97 (1.5)	0.62 (0.02)

Table 6. Performance metrics for various methods across different configurations.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Generative Flows on Discrete State Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design
Publication Status	Published
Publication Details	Campbell, A., Yim, J., Barzilay, R., Rainforth, T. and Jaakkola, T., 2024. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *International Conference on Machine Learning 5453–5512*

Student Confirmation

Student Name:	Andrew Campbell		
Contribution to the Paper	- Joint first author - Conceptualized and formalized the discrete and multimodal flow methodology with refinements from collaborators. - Wrote the code for and ran all of the experiments on text data. - Provided code contributions and jointly ran co-design experiments with collaborators. - Jointly wrote the manuscript with collaborators. - Conceptualized and proved all propositions.		
Signature		Date	08-Jan-2025

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Arnaud Doucet, Senior Staff Research Scientist Google DeepMind			
Supervisor comments I agree with the description of the contributions.			
Signature		Date	09-Jan-2025

This completed form should be included in the thesis, at the end of the relevant chapter.

7 Conclusions and Discussion

Contents

7.1 Conclusions

This thesis explored methods for performing generative modelling on complex data such as discrete data, trans-dimensional data, data lying on a manifold and multi-modal data.

In Chapter 1, we explored the problem of generative modelling from first principles. We understood the challenge in even defining the generative modelling problem and why complex probabilistic models are required to train networks to replicate and generalize the patterns observed in a dataset. By covering classical approaches to generative modelling through VAEs and GANs we discovered these pre-diffusion techniques fail to achieve all three of: high sample quality, good distributional coverage and stable training simultaneously. We then introduced the general concept of generative processes (of which diffusion models and flow-

based models are specific instances) on both continuous and discrete state spaces. Through their use of an iterative generative procedure, guided in some form by a fixed corruption process, we found generative processes can achieve all three generative modelling desiderata.

In Chapter 2, we reviewed existing work tackling the generative modelling and generic data problems. We covered the developments leading up to the popularization of diffusion models and correspondingly the development of flow-based models. We covered works expanding these techniques to discrete data, manifold data and multi-modal data as well as covering other generative techniques for these complex data types.

In Chapter 3, a formalism for discrete data diffusion modelling was introduced. Instead of using the discrete time hierarchical VAE style of diffusion model, the model is constructed through considering continuous time Markov chains (CTMCs) which led to more flexible sampling options. We derived a training objective and described efficient sampling methods for simulating the CTMCs at inference time enabling the method to achieve state of the art performance on high-dimensional discrete image modelling.

In Chapter 4, the task of modelling data with varying dimensionality using diffusion models was tackled. To enable the generative process to also generate the length of the datapoint, jumps were introduced into the diffusion process so that the model can add dimensions as necessary until a complete datapoint is created. We defined a model parameterization with a shared backbone to predict both a score and a jump component as well as training and sampling algorithms. The model enabled the generation of molecules and videos of varying dimensionality.

In Chapter 5, we introduced the use of flow-based models for the protein motif-scaffolding problem. The model is required to produce a protein structure that can hold in place a small number of pre-specified residues. An SE(3) flow model is used to perform the infilling by generating the required positions and rotations of scaffolding residues.

In Chapter 6, a multi-modal discrete continuous flow-based model was developed for the protein co-design task. We developed a framework for modelling discrete data using flows, again through the language of CTMCs. This framework was combined with continuous flows to create a generative model capable of simultaneously generating continuous protein structure jointly with the corresponding discrete amino acid sequence. Our approach was found to match the performance of the best two-stage single modality generation procedures whilst requiring only a single model.

7.2 Limitations

As described in Section 2.4.5, when applying diffusion or flow-based approaches for multi-modal data, a relative noise schedule between the modalities needs to be defined. This can affect the model performance as it implicitly defines an ordering in which modalities are generated. In Chapter 6, we dealt with this problem by learning a model over all relative levels of corruption between continuous structure and discrete sequence, enabling the choice of ordering to be made at inference time. However, this approach may not scale to many modalities in which case there is a combinatorial explosion of possible orderings. This leaves an open problem as to a principled approach for learning a suitable generation ordering. Any method needs to be influenced by the inductive biases of the neural network used because, theoretically speaking, all orderings are equally valid decompositions of the joint likelihood. The difference between the orderings will only become apparent if the learning problem for the generative network is easier under some orderings versus others.

In Chapter 4, our generative process includes jumps at which point an extra dimension can be created. This probabilistic model could be made more flexible by also allowing for deletions of dimensions during generation in case the model decides fewer dimensions are required. This addition would require careful changes to the corruption process because if the generative process includes deletions, the corruption process is required to include insertions. It is unclear exactly what

values should be inserted during corruption, previously deleted values could be used, however, this would result in a non-Markovian corruption process. Extending flow-based methods to the trans-dimensional problem may circumvent the need for Markovian corruption and enable the use of more flexible corruption processes.

Finally, we note there remains a disparity in performance between autoregressive models and discrete diffusion/flow models on the important modality of text. Due to their fixed left-to-right decomposition, autoregressive models can benefit from efficient training with causal masking in addition to the left-to-right ordering seeming a natural fit for text generation. In theory, the ability of a model to generate in any ordering can be beneficial for some tasks such as infilling, code generation or editing which naturally lend themselves to a parallelized back-and-forth reasoning style. It remains to be seen if a suitable corruption process and parameterization of discrete diffusion/flow models can be found to exploit these parallel structures and close the performance gap.

7.3 Extensions

The ideas in this thesis have subsequently been built-upon and refined by the research community. With regards to discrete data diffusion models, Sun et al. 2023; Benton et al. 2024; Lou et al. 2023 refine and simplify the training objective and sampling procedure. Zhao et al. 2024 build upon the observed improvement with the use of corrector steps by learning the corrector model whilst Wang et al. 2024 utilize the CTMC formalism to perform reinforcement learning based fine tuning of a discrete diffusion model. The CTMC approach has been used for generating diverse datasets such as graphs (Xu et al. 2024) and DNA (Nisonoff et al. 2024), the latter of which also introduced the use of guidance based approaches for CTMCs.

With respect to discrete flows, the design space of the methodology has been thoroughly explored by Gat et al. 2024 and the work of Shaul et al. 2024 deepens our theoretical understanding of discrete interpolants. The method has been

applied for graph generation by Qin et al. 2024 who make extensive use of the sampling time flexibility for improved generation performance whilst Hua et al. 2024 utilize the multi-modal flow framework to generate enzymes that are able to catalyse desired reactions.

7.4 Future Outlook

Although diffusion and flow models achieve high sample quality and good diversity, they are expensive to sample owing to the requirement of simulating an entire generative process which can necessitate hundreds of calls to the denoising network. In continuous spaces, there is a large body of work aiming to speed up the process through distillation and adversarial objectives as described in Section 2.1. The large simulation cost is also a problem in discrete spaces however it is less clear how speed ups through distillation can be achieved. This is because whereas a continuous ODE generative model defines a deterministic mapping between a continuous latent space to the data space, a discrete diffusion model is inherently stochastic. For example, a masking style generative model always begins generation at the same starting point, the all mask state, necessitating noise in the generative procedure to provide diversity in the sampling distribution. Procedures to distil a discrete trajectory would need to reparameterize the generative process and could also potentially revive the use of discrete adversarial objectives, as described in Section 2.4.2.

Another future direction is to build upon the dramatic performance improvement observed in Chapter 6 when the model was trained on a self-distillation dataset. This can be interpreted as a form of reinforcement learning whereby the model is aiming to more frequently sample high reward samples that have been discovered through previous sampling rounds of the model. The reward here being defined by a user specified utility function. The recipe of an initial generative modelling pre-training stage followed by reinforcement learning based fine-tuning has already proven successful for autoregressive models (Ouyang et al. 2022) and diffusion models (Wallace et al. 2024). These techniques have begun to be explored with

diffusion models on complex spaces (Li et al. 2024) whereby a value function is learned that guides the generative procedure towards higher reward samples. The possibility of these approaches, specifically in discrete spaces, could be further enhanced through taking other ideas from the reinforcement learning literature, such as Monte Carlo Tree Search (Silver et al. 2016). Powerful search strategies could be combined with an iterative procedure of re-training the generative model on discovered high reward samples, akin to iterative policy improvement (Sutton 2018), to enable highly desirable out-of-distribution samples to be created.

References

Albergo, Michael S, Nicholas M Boffi, and Eric Vanden-Eijnden. 2023a. "Stochastic interpolants: A unifying framework for flows and diffusions." *arXiv preprint arXiv:2303.08797.*

Albergo, Michael S, and Eric Vanden-Eijnden. 2023b. "Building normalizing flows with stochastic interpolants." *International Conference on Learning Representations.*

Anderson, Brian D O. 1982. "Reverse-time diffusion equation models." *Stochastic Process. Appl.*

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. "Wasserstein generative adversarial networks." *International conference on machine learning.*

Austin, Jacob, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. "Structured denoising diffusion models in discrete state-spaces." *Advances in Neural Information Processing Systems.*

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. 2013. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation." *arxiv.org/abs/1308.3432.*

Benton, Joe, Yuyang Shi, Valentin De Bortoli, George Deligiannidis, and Arnaud Doucet. 2024. "From denoising diffusions to denoising Markov models." *Journal of the Royal Statistical Society Series B: Statistical Methodology.*

Bowman, Samuel R, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2015. "Generating sentences from a continuous space." *arXiv preprint arXiv:1511.06349.*

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

Chen, Ricky T. Q., Brandon Amos, and Maximilian Nickel. 2021. "Neural Spatio-Temporal Point Processes." In *International Conference on Learning Representations.*

Chen, Ricky T. Q., and Yaron Lipman. 2024. "Flow Matching on General Geometries." In *The Twelfth International Conference on Learning Representations.*

Chen, Ting, Ruixiang Zhang, and Geoffrey Hinton. 2022. "Analog bits: Generating discrete data using diffusion models with self-conditioning." *arXiv preprint arXiv:2208.04202.*

Chen, Xi, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. 2018b. "Pixelsnail: An improved autoregressive generative model." In *International conference on machine learning.*

Cohen, Taco, and Max Welling. 2016. "Group equivariant convolutional networks." In *International conference on machine learning.*

De Bortoli, Valentin, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. 2022. "Riemannian Score-Based Generative Modelling." In *Advances in Neural Information Processing Systems,* edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh.

Del Moral, Pierre, and Spiridon Penev. 2017. *Stochastic Processes: From Applications to Theory.* Chapman / Hall/CRC.

Dey, Neel, Antong Chen, and Soheil Ghafurian. 2020. "Group equivariant generative adversarial networks." *arXiv preprint arXiv:2005.01683.*

Dieleman, Sander, Laurent Sartran, Arman Roshannai, Nikolay Savinov, Yaroslav Ganin, Pierre H Richemond, Arnaud Doucet, Robin Strudel, Chris Dyer, Conor Durkan, et al. 2022. "Continuous diffusion for categorical data." *arXiv preprint arXiv:2211.15089.*

Dinh, Laurent, David Krueger, and Yoshua Bengio. 2015. "NICE: Non-linear Independent Components Estimation." *arxiv.org/abs/1410.8516.*

Falorsi, Luca, Pim De Haan, Tim R Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S Cohen. 2018. "Explorations in homeomorphic variational auto-encoding." *arXiv preprint arXiv:1807.04689.*

Gao, Ruiqi, Emiel Hoogeboom, Jonathan Heek, Valentin De Bortoli, Kevin P. Murphy, and Tim Salimans. 2024. "Diffusion Meets Flow Matching: Two Sides of the Same Coin." https://diffusionflow.github.io/.

Gat, Itai, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. 2024. "Discrete flow matching." *arXiv preprint arXiv:2407.15595.*

Gebauer, Niklas, Michael Gastegger, and Kristof Schütt. 2019. "Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules." *Advances in neural information processing systems.*

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative adversarial nets." *Advances in neural information processing systems.*

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. "Improved training of wasserstein gans." *Advances in neural information processing systems.*

Gulrajani, Ishaan, and Tatsunori B Hashimoto. 2024. "Likelihood-based diffusion language models." *Advances in Neural Information Processing Systems.*

Hayes, Thomas, Roshan Rao, Halil Akin, Nicholas J Sofroniew, Deniz Oktay, Zeming Lin, Robert Verkuil, Vincent Q Tran, Jonathan Deaton, Marius Wiggert, et al. 2024. "Simulating 500 million years of evolution with a language model." *bioRxiv.*

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. 2020. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems.*

Hoogeboom, Emiel, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. 2021a. "Autoregressive diffusion models." *arXiv preprint arXiv:2110.02037.*

Hoogeboom, Emiel, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021b. "Argmax flows and multinomial diffusion: Learning categorical distributions." *Advances in Neural Information Processing Systems.*

Hoogeboom, Emiel, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. 2022. "Equivariant diffusion for molecule generation in 3d." *International Conference on Machine Learning.*

Hua, Chenqing, Yong Liu, Dinghuai Zhang, Odin Zhang, Sitao Luan, Kevin K Yang, Guy Wolf, Doina Precup, and Shuangjia Zheng. 2024. "Enzymeflow: Generating reaction-specific enzyme catalytic pockets through flow matching and co-evolutionary dynamics." *arXiv preprint arXiv:2410.00327.*

Hua, Chenqing, Sitao Luan, Minkai Xu, Zhitao Ying, Jie Fu, Stefano Ermon, and Doina Precup. 2023. "MUDiff: Unified Diffusion for Complete Molecule Generation." In *The Second Learning on Graphs Conference.*

Huang, Chin-Wei, Milad Aghajohari, Joey Bose, Prakash Panangaden, and Aaron C Courville. 2022. "Riemannian Diffusion Models." In *Advances in Neural Information Processing Systems,* edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh.

Huang, Chin-Wei, Jae Hyun Lim, and Aaron C Courville. 2021. "A variational perspective on diffusion-based generative models and score matching." *Advances in Neural Information Processing Systems.*

Hyvärinen, Aapo, and Peter Dayan. 2005. "Estimation of non-normalized statistical models by score matching." *Journal of Machine Learning Research.*

Jang, Eric, Shixiang Gu, and Ben Poole. 2017. "Categorical Reparameterization with Gumbel-Softmax." In *International Conference on Learning Representations.*

Karras, Tero, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. "Progressive Growing of GANs for Improved Quality, Stability, and Variation." In *International Conference on Learning Representations.*

Karras, Tero, Miika Aittala, Timo Aila, and Samuli Laine. 2022. "Elucidating the Design Space of Diffusion-Based Generative Models." *Advances in Neural Information Processing Systems.*

Kim, Dongjun, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. 2023. "Consistency trajectory models: Learning probability flow ode trajectory of diffusion." *arXiv preprint arXiv:2310.02279.*

Kingma, Diederik P, and Max Welling. 2013. "Auto-encoding variational bayes." *arXiv preprint arXiv:1312.6114.*

Klein, Leon, Andreas Krämer, and Frank Noe. 2023. "Equivariant flow matching." In *Thirty-seventh Conference on Neural Information Processing Systems.*

Klushyn, Alexej, Nutan Chen, Richard Kurle, Botond Cseke, and Patrick van der Smagt. 2019. "Learning hierarchical priors in vaes." *Advances in neural information processing systems.*

Li, Xiner, Yulai Zhao, Chenyu Wang, Gabriele Scalia, Gokcen Eraslan, Surag Nair, Tommaso Biancalani, Shuiwang Ji, Aviv Regev, Sergey Levine, et al. 2024. "Derivative-free guidance in continuous and discrete diffusion models with soft value-based decoding." *arXiv preprint arXiv:2408.08252.*

Lipman, Yaron, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. 2022. "Flow Matching for Generative Modeling." *arXiv preprint arXiv:2210.02747.*

Liu, Xingchao, Chengyue Gong, and Qiang Liu. 2022. "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow." *arXiv preprint arXiv:2209.03003.*

Lou, Aaron, Chenlin Meng, and Stefano Ermon. 2023. "Discrete diffusion language modeling by estimating the ratios of the data distribution." *International Conference on Machine Learning.*

Luhman, Eric, and Troy Luhman. 2021. "Knowledge distillation in iterative generative models for improved sampling speed." *arXiv preprint arXiv:2101.02388.*

Ma, Chao, Sebastian Tschiatschek, Richard Turner, José Miguel Hernández-Lobato, and Cheng Zhang. 2020. "VAEM: a deep generative model for heterogeneous mixed type data." *Advances in Neural Information Processing Systems.*

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. 2017. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables." In *International Conference on Learning Representations.*

Mathieu, Emile, and Maximilian Nickel. 2020. "Riemannian Continuous Normalizing Flows." In *Advances in Neural Information Processing Systems.*

Miao, Yishu, Lei Yu, and Phil Blunsom. 2016. "Neural variational inference for text processing." In *International conference on machine learning.*

Nisonoff, Hunter, Junhao Xiong, Stephan Allenspach, and Jennifer Listgarten. 2024. "Unlocking Guidance for Discrete State-Space Diffusion and Flow Models." *arXiv preprint arXiv:2406.01572.*

Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. "Training language models to follow instructions with human feedback." *Advances in neural information processing systems.*

Pannatier, Arnaud, Evann Courdier, and François Fleuret. 2024. "σ-GPTs: A New Approach to Autoregressive Models." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.*

Peluchetti, Stefano. 2021. "Non-denoising forward-time diffusions." *unpublished.*

Peng, Xingang, Jiaqi Guan, Qiang Liu, and Jianzhu Ma. 2023. "MolDiff: Addressing the Atom-Bond Inconsistency Problem in 3D Molecule Diffusion Generation." In *Proceedings of the 40th International Conference on Machine Learning.*

Qin, Yiming, Manuel Madeira, Dorina Thanou, and Pascal Frossard. 2024. "DeFoG: Discrete Flow Matching for Graph Generation." *arXiv preprint arXiv:2410.04263.*

Rezende, Danilo, and Shakir Mohamed. 2015. "Variational inference with normalizing flows." In *International conference on machine learning.*

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. "U-net: Convolutional networks for biomedical image segmentation." *Medical image computing and computer-assisted intervention.*

Salimans, Tim, and Jonathan Ho. 2022. "Progressive distillation for fast sampling of diffusion models." *arXiv preprint arXiv:2202.00512.*

Särkkä, Simo, and Arno Solin. 2019. "Applied Stochastic Differential Equations." *Cambridge University Press.*

Sauer, Axel, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. 2025. "Adversarial diffusion distillation." In *European Conference on Computer Vision.*

Shaul, Neta, Itai Gat, Marton Havasi, Daniel Severo, Anuroop Sriram, Peter Holderrieth, Brian Karrer, Yaron Lipman, and Ricky TQ Chen. 2024. "Flow Matching with General Discrete Paths: A Kinetic-Optimal Perspective." *arXiv preprint arXiv:2412.03487.*

Shi, Yuyang, Valentin De Bortoli, Andrew Campbell, and Arnaud Doucet. 2024. "Diffusion Schrödinger bridge matching." *Advances in Neural Information Processing Systems.*

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. "Mastering the game of Go with deep neural networks and tree search." *nature.*

Sohl-Dickstein, Jascha, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning.*

Song, Yang, and Prafulla Dhariwal. 2023a. "Improved techniques for training consistency models." *arXiv preprint arXiv:2310.14189.*

Song, Yang, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. 2023b. "Consistency models." *arXiv preprint arXiv:2303.01469.*

Song, Yang, Conor Durkan, Iain Murray, and Stefano Ermon. 2021a. "Maximum likelihood training of score-based diffusion models." *Advances in neural information processing systems.*

Song, Yang, and Stefano Ermon. 2019. "Generative modeling by estimating gradients of the data distribution." *Advances in Neural Information Processing Systems.*

Song, Yang, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021b. "Score-based generative modeling through stochastic differential equations." *International Conference on Learning Representations.*

Sun, Haoran, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. 2023. "Score-based Continuous-time Discrete Diffusion Models." In *The Eleventh International Conference on Learning Representations.*

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. "Sequence to Sequence Learning with Neural Networks." *arxiv.org/abs/1409.3215.*

Sutton, Richard S. 2018. "Reinforcement learning: An introduction." *A Bradford Book.*

Theis, Lucas, and Matthias Bethge. 2015. "Generative image modeling using spatial lstms." *Advances in neural information processing systems.*

Tomczak, Jakub, and Max Welling. 2018. "VAE with a VampPrior." *International conference on artificial intelligence and statistics.*

Tran, Dustin, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. 2019. "Discrete Flows: Invertible Generative Models of Discrete Data." In *Advances in Neural Information Processing Systems,* edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett.

Uria, Benigno, Iain Murray, and Hugo Larochelle. 2013. "RNADE: The real-valued neural autoregressive density-estimator." *Advances in Neural Information Processing Systems.*

Vahdat, Arash, and Jan Kautz. 2020. "NVAE: A deep hierarchical variational autoencoder." *Advances in neural information processing systems.*

Van Den Oord, Aaron, Oriol Vinyals, et al. 2017. "Neural discrete representation learning." *Advances in neural information processing systems.*

Vignac, Clement, Nagham Osman, Laura Toni, and Pascal Frossard. 2023. "MiDi: Mixed Graph and 3D Denoising Diffusion for Molecule Generation." In *ICLR 2023 - Machine Learning for Drug Discovery workshop.*

Vincent, Pascal. 2011. "A connection between score matching and denoising autoencoders." *Neural computation.*

Wallace, Bram, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. 2024. "Diffusion model alignment using direct preference optimization." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

Wang, Chenyu, Masatoshi Uehara, Yichun He, Amy Wang, Tommaso Biancalani, Avantika Lal, Tommi Jaakkola, Sergey Levine, Hanchen Wang, and Aviv Regev. 2024. "Fine-tuning discrete diffusion models via reward optimization with applications to dna and protein design." *arXiv preprint arXiv:2410.13643.*

Wildberger, Jonas, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf. 2023. "Flow matching for scalable simulation-based inference." *Advances in Neural Information Processing Systems* 36:16837–16864.

Williams, Ronald J. 1992. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Machine learning.*

Xu, Zhe, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. 2024. "Discrete-state Continuous-time Diffusion for Graph Generation." *arXiv preprint arXiv:2405.11416.*

Yang, Zhilin. 2019. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." *arXiv preprint arXiv:1906.08237.*

Yu, Lantao, Weinan Zhang, Jun Wang, and Yong Yu. 2017. "Seqgan: Sequence generative adversarial nets with policy gradient." In *Proceedings of the AAAI conference on artificial intelligence.*

Zhao, Yixiu, Jiaxin Shi, Lester Mackey, and Scott Linderman. 2024. "Informed correctors for discrete diffusion models." *arXiv preprint arXiv:2407.21243.*

Zheng, Qinqing, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky T. Q. Chen. 2023. "Guided Flows for Generative Modeling and Decision Making." *arxiv.org/abs/2311.13443.*

Zhou, Chunting, Lili Yu, Arun Babu, Kushal Tirumala, Michihiro Yasunaga, Leonid Shamis, Jacob Kahn, Xuezhe Ma, Luke Zettlemoyer, and Omer Levy. 2024. "Transfusion: Predict the next token and diffuse images with one multi-modal model." *arXiv preprint arXiv:2408.11039.*