University of Oxford

Different Conditional Simulation Techniques in Diffusion Models

by

Harry Amad

Oriel College

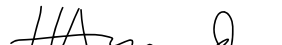
A dissertation submitted in partial fulfilment of the degree of Master of Science in Statistical Science.

Department of Statistics, 24–29 St Giles,
Oxford, OX1 3LB

September 2023

This is my own work (except where otherwise indicated).

Candidate: Harry Amad

Signed: 

Date:11/09/2023

Abstract

Diffusion models have recently emerged as a leading family of generative models, with superior performance in a variety of domains, both unconditionally and conditionally. In this dissertation, we introduce the fundamental elements of diffusion models, before exploring three techniques for conditional simulation. We investigate the theoretical underpinnings of tailored conditional diffusion models, as well as techniques which augment unconditional models to incorporate conditional information, namely the replacement and SMCDiff methods. We then experimentally compare these three techniques on a simple 2D dataset, a more complicated Bayesian inference problem, and an image inpainting task using the MNIST dataset, to assess how they perform in practice. We then discuss which of these methods is preferable in a broad set of scenarios.

Acknowledgements

I would firstly like to thank my supervisor Hai-Dang Dau for investing so much of his time to provide excellent guidance throughout the course of this project. I would also like to express my gratitude to my family and girlfriend whose support was invaluable, and especially to my parents, without whom my studies at Oxford would not be possible. Finally, to my dog Roy, I hope you have found the most comfortable spot by the eternal fireplace.

Contents

List of Figures

List of Tables

1 Introduction

Deep generative models encompass a wide variety of powerful methods that can learn arbitrary data distributions from a training dataset, so that synthetic samples can be drawn from the underlying distribution. The concept of generative models is not a new one, as learning complex probability distributions, especially those that describe real-world data, is a problem central to machine learning. However, interest in deep generative models in particular has been driven recently due to improvements in deep learning architectures and the increasing availability of large datasets for training [Bond-Taylor et al., 2022]. Various approaches have enjoyed a flurry of research over the past decade, including variational autoencoders (VAEs) [Kingma and Welling, 2022], generative adversarial networks (GANs) [Goodfellow et al., 2014], normalizing flows [Rezende and Mohamed, 2016], and, most recently, diffusion models [Sohl-Dickstein et al., 2015, Ho et al., 2020, Song and Ermon, 2020, Song et al., 2021]. These approaches can produce impressive state-of-the-art samples in a number of domains, including image generation (Figure 1), text-to-image (Figure 2), protein synthesis (Figure 3), audio generation, video generation, and many more.

Diffusion models in particular have recently emerged as one of the top performing deep generative modelling techniques. Initially applied to image synthesis, diffusion models have overtaken the previously dominant GANs in this popular domain [Dhariwal and Nichol, 2021], and they currently demonstrate state-of-the-art performance across multiple domains [Yang et al., 2023]. In this chapter we will give a brief introduction to the underpinnings of diffusion models and their three main frameworks—stochastic differential equation diffusion models (SDEs)[Song et al., 2021], denoising diffusion probabilistic models (DDPMs)[Sohl-Dickstein et al., 2015, Ho et al., 2020], and score-based generative models (SGMs)[Song and Ermon, 2020].

Figure 1: Images produced in Karras et al. [2019] by StyleGAN trained on the FFHQ dataset.

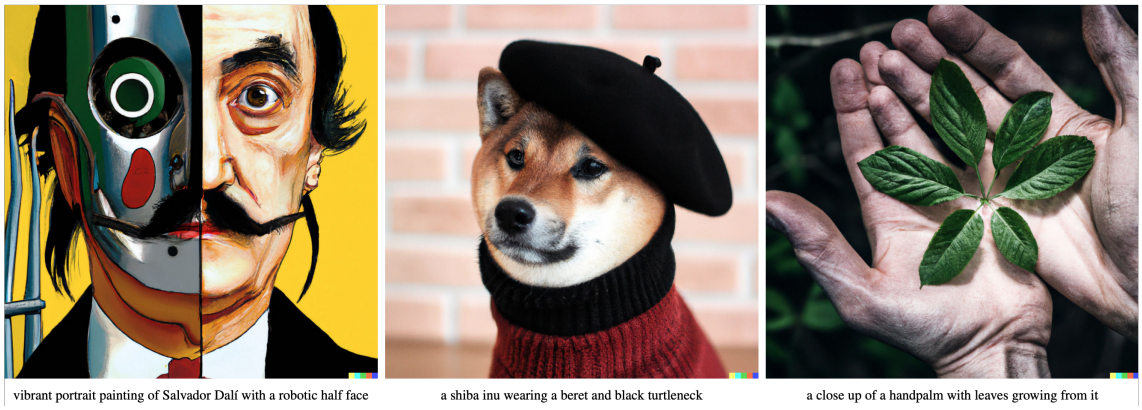


Figure 2: 1024x1024 image samples from Ramesh et al. [2022] produced by a diffusion model conditioned on the associated text prompt.

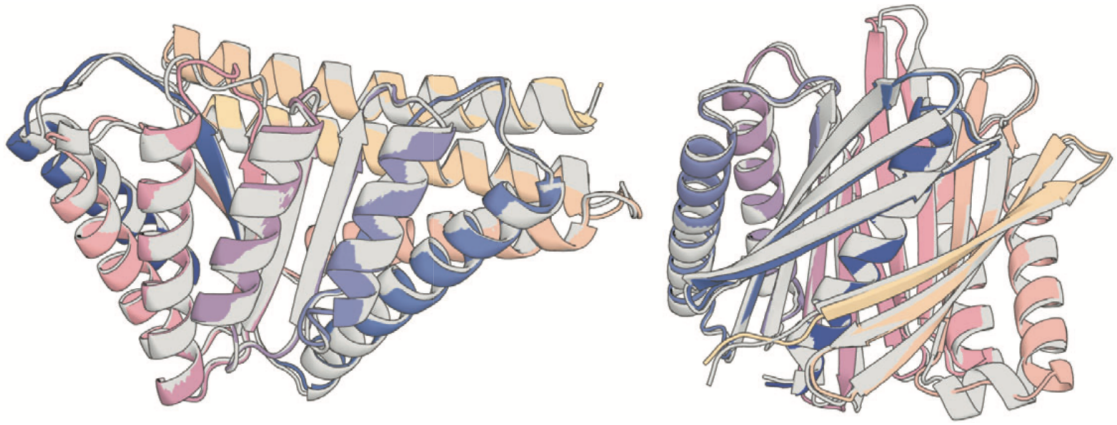


Figure 3: Proteins of length up to 400 amino acids can be generated by RFdiffusion in Watson et al. [2023].

1.1 Diffusion Models

The family of diffusion models share an overarching approach to generating data—noise is progressively added to training data and a neural network learns to reverse this process to generate data from noise. Diffusion models incrementally inject noise to training data until it resembles some known reference distribution p_{ref}—such as a standard Gaussian—in a procedure known as the 'forward process'. The model then learns to reverse this noising process with a neural network, so that realistic data can be generated by initially sampling from p_{ref}, and then progressively denoising the samples through the 'reverse process'. A simple example which demonstrates the intuition behind this process is shown in Figure 4 on a toy dataset. Most implementations of diffusion models in current research can be categorised into one of three frameworks: SDEs, DDPMs, or SGMs. Each of these conduct the noising and denoising of data in different ways, which we will now explain.

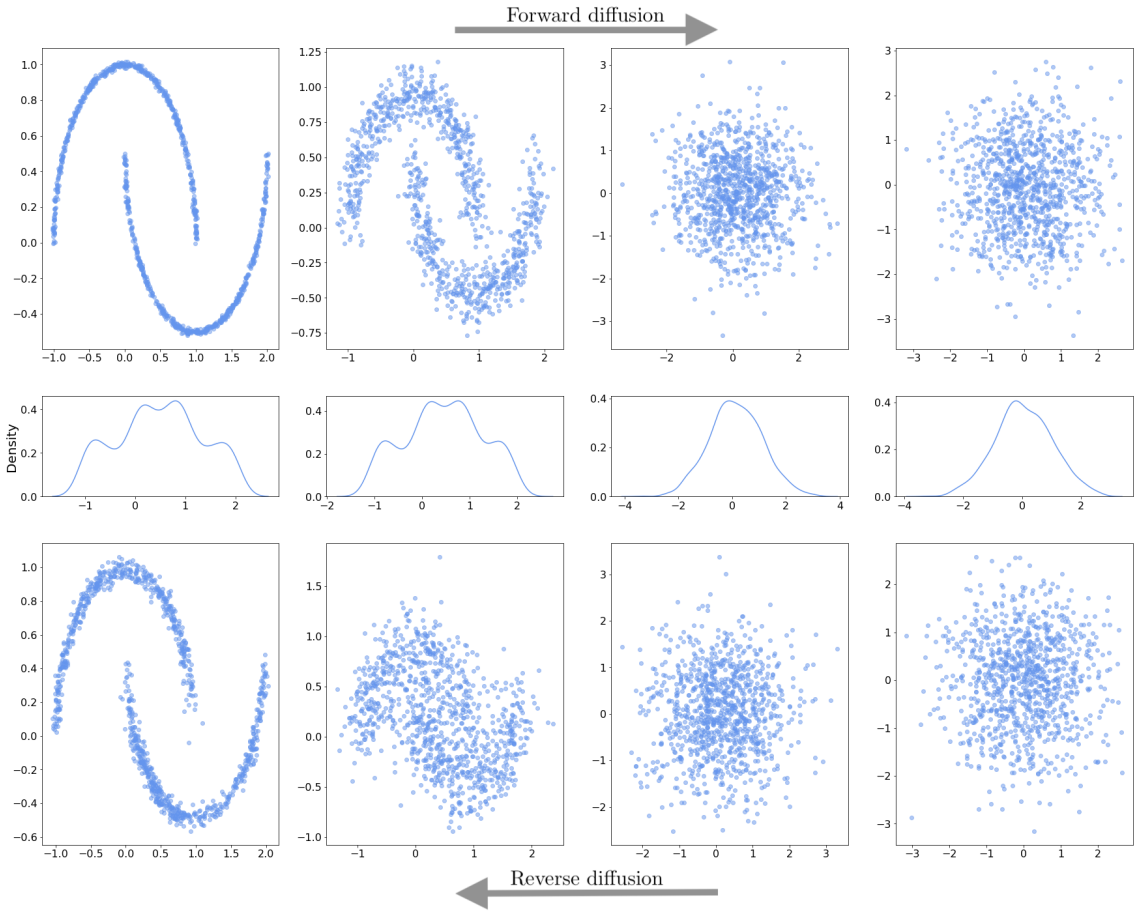


Figure 4: Basic diffusion model. On the top row, the original data is gradually perturbed by the forward process until it resembles a tractable reference distribution (in this case a standard 2D Gaussian). The middle row plots the marginal density of the x-axis variable. On the bottom row, the reverse process begins with data from the reference distribution and slowly adds structure until an approximation of the original data distribution is reached.

1.1.1 Stochastic Differential Equations

The SDE diffusion model framework is the most recent and complete formulation, and it is the foundation upon which the majority of recent diffusion models are built. SDE diffusion was proposed by Song et al. [2021] and describes the forward process with an arbitrary SDE, from which the reverse process can be derived using the score of the marginal probability density which in turn is estimated with a time-dependant neural network. Once the reverse SDE has been approximated, samples can be drawn from the data distribution p_{data} by first sampling from the prior p_{ref} and then solving the reverse SDE with these starting points, which can be done with a variety of numerical SDE and deterministic ordinary differential equation (ODE) solvers.

Formally, a diffusion process $\{x(t)\}_{t=0}^{T}$ with continuous time variable $t \in [0, T]$ is constructed where $x(0) \sim p_{\text{data}}$ and $x(T)$ is from a tractable prior p_{ref}. This process is governed by an Itô SDE

$$d\mathbf{x} = f(x, t)dt + g(t)d\mathbf{w} \tag{1}$$

where $\mathbf{w}$ is the standard Wiener process, f is the drift coefficient and g is the diffusion coefficient. Importantly, the reverse of a diffusion process is also a diffusion process with time running backwards [Anderson, 1982], a result that we will now explain the intuition behind. Consider discretizing the forward SDE (1) into N timesteps with stepsize Δt, giving rise to a Markov chain $\{x_t\}_{t=0}^{N}$ of perturbed data. The transition kernel $p_{t+1|t}$ can be defined as

$$p_{t+1|t}(x_{t+1}|x_t) = \mathcal{N}(x_{t+1}; x_t + \Delta t\ f_t, \Delta t\ g_t^2 I) \tag{2}$$

where we write $f(x_t, t) = f_t$ and $g(t) = g_t$ for brevity. We can determine the reverse transition kernel $p_{t|t+1}$ using Bayes' rule, resulting in

$$p_{t|t+1}(x_t|x_{t+1}) = \frac{p_t(x_t)p_{t+1|t}(x_{t+1}|x_t)}{p_{t+1}(x_{t+1})} \tag{3}$$

We can approximate this reverse transition kernel as follows

$$\begin{aligned} p_{t|t+1}(x_t|x_{t+1}) &= p_{t+1|t}(x_{t+1}|x_t)\exp[\log p_t(x_t) - \log p_{t+1}(x_{t+1})] && (4)\\ &\approx \mathcal{N}(x_t; x_{t+1} - \Delta t(f_{t+1} + g_{t+1}^2 \nabla \log p_{t+1}(x_{t+1})), \Delta t\ g_{t+1}^2 I) && (5) \end{aligned}$$

where the approximation in the second line is made using a Taylor expansion of $\log p_{t+1}$ at x_{t+1}, and the fact that $p_t \approx p_{t+1}$, $f_t \approx f_{t+1}$ when Δt is small. Now, as Δt becomes an infinitely small timestep, we can write the continuous generalization of this reverse transition kernel as the SDE

$$d\mathbf{x} = [f(x, t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)d\bar{\mathbf{w}} \tag{6}$$

where $\bar{\mathbf{w}}$ is the standard Wiener process with time flowing backwards, and dt is a negative timestep. Anderson [1982] provide a rigorous proof of this result, but nevertheless obtain the same reverse SDE as we do in (6), which we can use to produce samples from p_{data}. While we cannot obtain $\nabla_x \log p_t(x)$ in general, we can estimate it using a time-dependant neural network with a variety of score matching techniques, such as denoising score matching [Vincent, 2011], sliced score matching [Song et al., 2019], or finite-difference score matching [Pang et al., 2020].

Denoising score matching is used in Song et al. [2021], which we will describe the rationale behind, as is done in De Bortoli et al. [2023]. Consider the same discretization

of the forward process as described above. We can obtain the conditional density $p_{t+1|0}(x_{t+1}|x_0)$ because of the properties of the Gaussian distribution. Specifically, we can write

$$x_{t+1} = x_t + \Delta t\ f_t + \sqrt{\Delta t}\ g_t\ Z_t,\ \ Z_i \sim \mathcal{N}(0,1) \tag{7}$$
$$= x_{t-1} + \Delta t\ f_{t-1} + \sqrt{\Delta t}\ g_{t-1}\ Z_{t-1} + \Delta t\ f_t + \sqrt{\Delta t}\ g_t\ Z_t \tag{8}$$
$$= x_0 + \Delta t(f_0 + ... + f_{t-1} + f_t) + \sqrt{\Delta t}(g_0 Z_0 + ... + g_{t-1} Z_{t-1} + g_t Z_t) \tag{9}$$

which is a sum of Gaussians. Therefore we can obtain $p_{t+1|0}(x_{t+1}|x_0)$ which will be a Gaussian. We then have $p_{t+1}(x_{t+1}) = \int p_0(x_0) p_{t+1|0}(x_{t+1}|x_0) dx_0$, and $\nabla \log p_{t+1}(x_{t+1}) = \mathbb{E}_{p_{0|k+1}}[\nabla_{x_{t+1}} \log p_{k+1|0}(x_{k+1}|X_0)]$. We can therefore consider score matching as a regression problem, and use a neural network $s_\theta(x_t, t)$ as a functional approximator to learn $\nabla_x \log p_t(x)$, optimizing θ with the objective

$$\sum_{t=1}^{N} \mathbb{E}_{p_{0,t}}[||s_\theta(x_t, t) - \nabla_{x_t} \log p_{t|0}(x_t|x_0)||^2] \tag{10}$$

This intuition transfers to the continuous definition of the forward process, and Song et al. [2021] define a time-dependant neural network $s_\theta(x(t), t)$ to approximate $\nabla_x \log p_t(x)$ using the objective

$$\mathbb{E}_{t\sim\mathcal{U}(0,T), x(0)\sim p_{\text{data}}(x), x(t)\sim p(x(t)|x(0))}[\lambda(t)||s_\theta(x(t), t) - \nabla_{x(t)} \log p(x(t)|x(0))||^2] \tag{11}$$

where $\lambda(t)$ is a positive weighting function. To optimise the objective (11) we need to be able to sample from the transition kernels $p(x(t)|x(0))$ for all t to calculate the expectation using Monte Carlo methods. When the drift coefficient f is affine, the transition kernel will be a Gaussian distribution, which we have showed the intuition for in (7). This distribution can be found with standard techniques such as those in Section 5.5 of Särkkä and Solin [2019]. For general SDEs where f is not necessarily affine, Kolmonogorov's forward equation [Øksendal, 2010] can be solved to obtain the transition kernel.

While the forward diffusion can be defined as any SDE, Song et al. [2021] provide some concrete examples of SDEs based on the forward processes from the DDPM and SGM frameworks. While we have not yet covered these frameworks, all that we need to know now is that they approach the forward and reverse diffusion processes in a discrete manner, and Song et al. [2021] propose their continuous generalisations as potential forward SDE definitions. We will go into more detail on the rationales behind the forward processes in their respective sections.

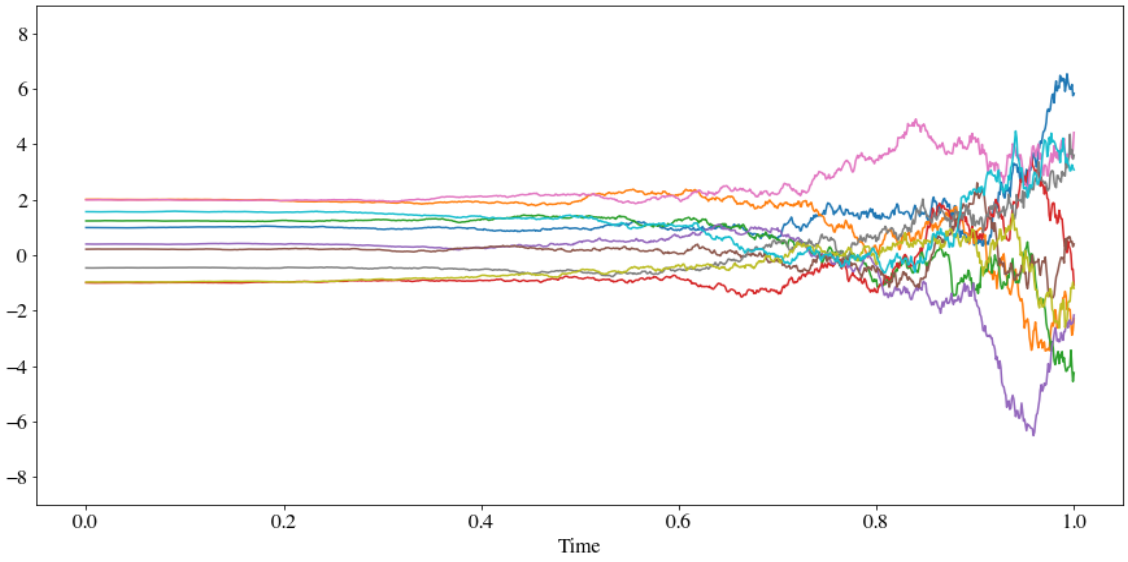


Figure 5: VE SDE forward diffusion process.

Given an increasing function $\sigma(t)$ for $t \in [0, 1]$, the SDE derived from the SGM framework is

$$d\mathbf{x} = \sqrt{\frac{d[\sigma^2(t)]}{dt}} d\mathbf{w} \tag{12}$$

Song et al. [2021] name this SDE the Variance Exploding (VE) SDE, as the variance of the process increases with t, and the resulting reference distribution is $p_{\text{ref}} = \mathcal{N}(0, \sigma(1)^2 I)$. This exploding variance nature can be seen in Figure 5 which shows the perturbation by the VE SDE of data taken from the original distribution used in Figure 4. For the first half of the forward process the noised points are relatively close to their original locations, and the variance of the points is similar to the original distribution variance. The variance of the points clearly increases from there, until all information from the original data distribution is lost when $t = 1$.

For the SDE derived from the DDPM framework, we define an increasing function $\beta(t) \in (0, 1)$ for $t \in [0, 1]$, and the resulting SDE is

$$d\mathbf{x} = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}d\mathbf{w} \tag{13}$$

Unlike the VE SDE, this SDE gives a stochastic process with a bounded unit variance when the original distribution also has unit variance, and results in the reference distribution $p_{\text{ref}} = \mathcal{N}(0, I)$. As such, Song et al. [2021] name this the Variance Preserving (VP) SDE. This behaviour is shown in Figure 6, where data from the original distribution in Figure 4 is perturbed by the VP SDE. We can see that the variance of the process is stable, in clear contrast to Figure 5.

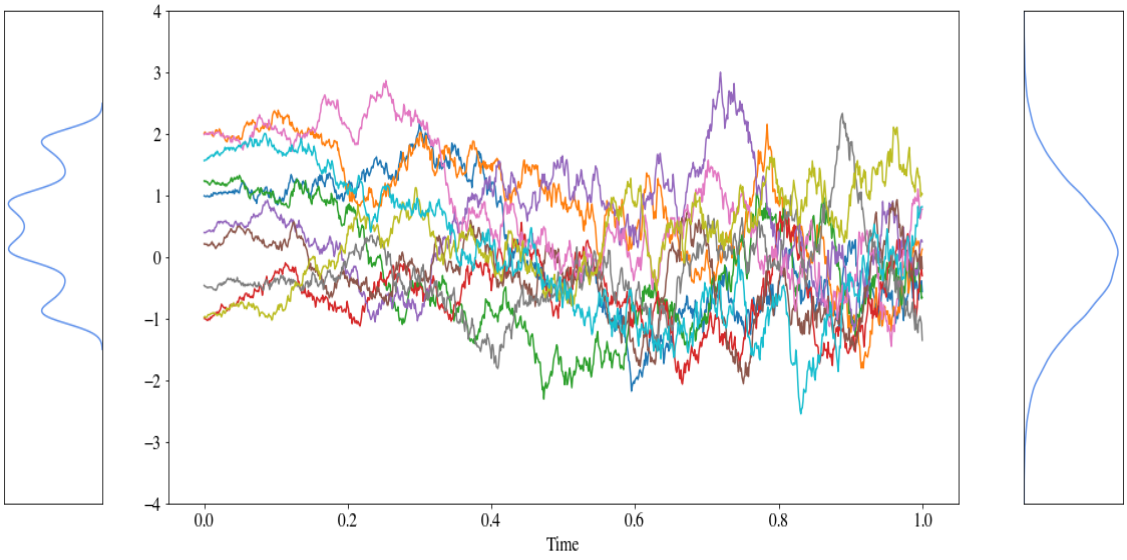


Figure 6: VP SDE forward diffusion process. On the left is the density of the original data disribution. On the right is the prior density, $\mathcal{N}(0, I)$.

While Song et al. [2021] show that the VE and VP SDEs perform well on image generation tasks, they are by no means the only options for the forward process SDE. The SDE framework allows for a flexible definition of the forward process, which can be tailored to whatever works best for a specific dataset.

Once the forward SDE has been defined and a time-dependent neural network has been trained using objective (11), we can approximate the reverse time SDE by substituting the neural network's functional approximation of $\nabla_x \log p_t(x)$ into (6), giving the approximation

$$d\mathbf{x} = [f(x,t) - g(t)^2 s_\theta(x,t)]dt + g(t)d\bar{\mathbf{w}} \tag{14}$$

This reverse SDE can be simulated with numerical SDE solvers to draw approximate samples from p_{data}. There are multiple options for this which discretize the stochastic process differently, such as the Euler-Maruyama and stochastic Runge-Kutta methods [Kloeden and Platen, 2011]. The Euler-Maruyama discretization is used in Song et al. [2021] and also in our experiments in Section 3, which we will describe here. It approximates the true solution of the reverse SDE using a similar discretization to what we describe earlier in this section when outlining the intuition for the proof in Anderson [1982]. The time interval $t \in [0, T]$ is partitioned into n even sub-intervals of width $\Delta t = \frac{T}{n}$. x_T is sampled from p_{ref}, and then iteratively updated as described by the equation

$$x_{t-1} = x_t - \Delta t(f(x,t) - g(t)^2 s_\theta(x,t)) + g(t)\sqrt{\Delta t}\ Z_t,\ \ Z_t \sim \mathcal{N}(0,1) \tag{15}$$

until x_0 is reached.

However, this sampling approach does not make full use of the information made available by the neural network $s_\theta(x,t) \approx \nabla_x \log p_t(x)$, and we can augment its samples using score-based MCMC approaches, such as Langevin MCMC [Parisi, 1981, Grenander and Miller, 1994] or Hamiltonian Monte Carlo [Brooks et al., 2011]. At each time step, once the numerical solver—the 'predictor'—gives an estimate of the sample for the next step, the score-based MCMC approach—the 'corrector'—can correct the marginal distribution of this estimate. This sampling method is hence named Predictor-Corrector (PC) sampling, and Song et al. [2021] show that it empirically improves upon samples from numerical solvers, although with increased computation.

Song et al. [2021] also propose a sampling method using ODE solvers. For a given diffusion process, there exists a deterministic process which shares the same marginal probability densities $\{p_t(x)\}_{t=0}^{T}$ as the SDE, given by the ODE

$$d\mathbf{x} = [f(x,t) - \frac{1}{2} g(t)^2 \nabla_x \log p_t(x)]dt \tag{16}$$

which is named the probability flow ODE. This sampling method offers a fast way of generating samples using black-box ODE solvers [Dormand and Prince, 1980], which Song et al. [2021] show can generate image samples of similar visual quality to the previously mentioned methods while reducing the number of function evaluations by over 90%. Furthermore, this formulation presents some opportunities not available when using a numerical SDE solver or PC sampling.

The probability flow ODE allows for exact likelihood computation, since the density defined by (16) can be found using instantaneous change of variables as done in Chen et al. [2019]. This density can then be used to compute the likelihood of any data, which can be useful to assess the likelihood of the model on a test set from the same distribution as the training data. Furthermore, integration of (16) allows the encoding of a datapoint $x(0)$ into the latent space $x(T)$. Manipulation of these latent representations directly impacts the data $x(0)$. As such, latent representations can be adapted for tasks such as interpolation for image editing.

We will now discuss the DDPM and SGM frameworks, which take a discrete approach to the the forward and reverse processes of diffusion models. Both were proposed prior to the SDE framework, and so this will provide historical context, as well as the intuition behind the VE and VP SDEs we introduced earlier.

1.1.2 Denoising Diffusion Probabilistic Models

DDPM were first introduced in Sohl-Dickstein et al. [2015] as a novel generative model, and then expanded upon and significantly improved in Ho et al. [2020], achieving record performance on the CIFAR10 image dataset. While this framework was proposed prior to the SDE framework, we will consider it as a particular

discretization of the VP SDE from (13). DDPMs are defined by two finite Markov chains for the forward and reverse processes, which are discretizations of the respective processes from the SDE framework.

The forward diffusion gradually adds noise to data over some number of steps T to convert data x_0 from any complex distribution to a tractable prior at x_T. The transition kernels of the forward Markov process are designed to ensure that data is transformed to this simple prior distribution—typically the standard Gaussian. The second Markov chain defines the reverse process, where transition kernels are learnt to undo the noising done by the forward diffusion, to convert noise to data. These transition kernels are parameterized by a time-dependent neural network used to predict the next step in the reverse process x_{t-1} based on the current, noisy input x_t. Therefore, data can be generated by first drawing samples x_T from the prior and then sampling from the reverse transition kernels until x_0 is reached.

More formally, defining the reference distribution p_{ref} as the standard Gaussian, consider d-dimensional training samples from some distribution q_0 on $\mathbb{R}^d$. The forward process can be defined as a Markov chain of length T with joint density

$$q(x_{0:T}) = q_0(x_0) \prod_{t=1}^{T} q_{t|t-1}(x_t|x_{t-1}) \tag{17}$$

Some monotonically increasing variance schedule $\beta_1, ..., \beta_T$ with $\beta_t \in (0, 1)$ is the corresponding discretization of $\beta(t)$ from (13), and the transition kernels $q_{t|t-1}$ are defined as

$$q_{t|t-1}(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{18}$$

A notable feature of this design is that, because of the properties of the Gaussian distribution, samples x_t from any timestep t can be drawn in constant time given x_0, since

$$\alpha_t := 1 - \beta_t, \;\; \bar{\alpha}_t := \prod_{i=1}^{t} \alpha_i \tag{19}$$

$$q_{t|0}(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}}_t x_0, (1-\bar{\alpha}_t)I) \tag{20}$$

following the intuition shown in (7). This property is critical as it allows quick sampling from any arbitrary timestep in the forward process, which is necessary to ensure that the training of the reverse process can be done in a reasonable amount of time. Also, since $\{\beta_t\}_{t=1}^{T}$ is monotonically increasing with t, α_t will shrink as t grows and $\bar{\alpha}_T \approx 0$ if the forward process is run for enough timesteps. Therefore, for large

T and considering (20) with $t = T$, $q_{T|0}(x_T|x_0) \approx \mathcal{N}(x_T; 0, I)$ and the distribution of x_T is

$$q_T(x_T) = \int q_{T|0}(x_T|x_0)q_0(x_0)dx_0 \approx \mathcal{N}(x_T; 0, I) \tag{21}$$

which is approximately our desired reference distribution p_{ref}.

When β_t is small, the forward process approximates a continuous diffusion [Sohl-Dickstein et al., 2015] similar to the SDE framework. The larger the number of timesteps T, the smaller we can allow β_t to be while still having $\bar{\alpha}_T \approx 0$, so with sufficiently large T we can assume the forward process is a continuous diffusion. Therefore, it is appropriate to approach the reverse process as a discretisation of the reverse process of the VP SDE. As such, our approximation of the reverse process Markov chain $\{x_t\}_{t=0}^T$ has joint density

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{22}$$

with the following prior distribution and transition kernels

$$p(x_T) = \mathcal{N}(x_T; 0, I) \tag{23}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta_t I) \tag{24}$$

The mean of the transition kernels, $\mu_\theta(x_t, t)$, is parameterized by a neural network where θ denotes the network parameters that are optimized in training.

To train the reverse process, Ho et al. [2020] adopt a different approach than Song et al. [2021]. Rather than considering score matching techniques, optimization is done to minimize the KL divergence between $q(x_{0:T})$ and $p_\theta(x_{0:T})$, or equivalently by minimising the variational bound on negative log-likelihood L_{vb}, where

$$L_{\text{vb}} := \mathbb{E}_{q(x_{0:T})}[-\log p_\theta(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}] \tag{25}$$

Ho et al. [2020] show that minimising (25) is equivalent to minimising the objective

$$\mathbb{E}_{t\sim\mathcal{U}\{1,T\},x_0\sim q_0,\epsilon\sim\mathcal{N}(0,I)}[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2] \tag{26}$$

where $\epsilon \sim \mathcal{N}(0, I)$ is the noise added to x_0 to get x_t by the forward process, ϵ_θ denotes the neural network intended to predict this noise, and t is from the discrete uniform

distribution between 1 and T. Furthermore, they propose an alternate training objective by reweighting (26) to introduce inductive biases into the training process that improve sample quality. The proposed alternate objective is

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t\sim\mathcal{U}\{1,T\},x_0\sim q_0,\epsilon\sim\mathcal{N}(0,I)}[||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2] \tag{27}$$

This proposed objective down-weights loss terms with small t, where the noise is minimal, and incentives the model to optimize for denoising when t is large, empirically producing better samples. It also is an exceptionally simple loss to implement practically, as it is just the mean-squared error between the output of the neural network and the actual noise added at time t.

When the reverse process is trained with objective (27), sampling can be done by first drawing samples x_T from the Gaussian prior distribution, and then at each timestep predicting the noise $\epsilon_\theta(x_t, t)$ and removing it from x_t to predict $\hat{x_0}$. Then, x_{t-1} is derived using $p_{t-1|0}(x_{t-1}|\hat{x_0})$ from (20). This is done until x_0 is reached, as described by Ho et al. [2020] in Algorithm 1.

Algorithm 1 Sampling with the DDPM framework

$x_T \sim \mathcal{N}(0, I)$
for $t = T, ..., 1$ **do**
  $z \sim \mathcal{N}(z; 0, I)$ if $t > 1$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sigma_t z$
end for
return x_0

1.1.3 Score Based Generative Models

SGMs were introduced by Song and Ermon [2020], and then improved in high dimensional spaces in Song and Ermon [2020]. The intuition behind SGMs is to learn the score of a probability density $\nabla_x \log p(x)$ which can then be used to generate data that follows $p(x)$. However, there are two main challenges that prevent a naïve implementation of SGM from working. Firstly, the manifold hypothesis states that the data encountered in the real world often lies on a low dimensional manifold embedded in a higher dimensional space [Belkin and Niyogi, 2003]. Considering this hypothesis, naïve SGMs struggle because $\nabla_x \log p(x)$ is undefined when x is confined to a low dimensional manifold. Furthermore, low density regions of $p(x)$ will result in training data from that region being scarce, which causes difficulties for estimating $\nabla_x \log p(x)$ in these regions. However, perturbation of the training data can make learning the score of a distribution more manageable. Perturbing data with Gaussian noise will ensure that the data is not confined to a low dimensional manifold. Furthermore, large Gaussian noise compared to the scale of the original data distribution will ensure that low density regions are filled in. With these additions, SGMs become a viable generative process.

Like DDPM, while SGM were proposed before SDE diffusion models, we will consider them as a discretization of the VE SDE in (12) over L timesteps. Data is corrupted with perturbation kernels $p_\sigma(\tilde{x}|x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I)$ with increasing noise levels $0 < \sigma_1 < ... < \sigma_L$ which are a discretization of $\sigma(t)$ in (12). Defining the distribution of the noised data as $p_\sigma(\tilde{x}) = \int p_{\text{data}}(x) p_\sigma(\tilde{x}|x) dx$ where p_{data} is the original data distribution, σ_1 is chosen to be small, such that $p_{\sigma_1}(x) \approx p_{\text{data}}(x)$, and σ_T is chosen to be large enough that $p_{\sigma_T}(x) \approx \mathcal{N}(0, \sigma_L^2 I)$. A neural network $s_\theta(\tilde{x}, \sigma_i)$ can then be trained to learn the score function, $\nabla_{\tilde{x}} \log p_{\sigma_i}(\tilde{x}|x)$, for the noised data when conditioned on the noise level σ_i. Similar to the SDE framework, denoising score matching is used in Song and Ermon [2020], leading to the objective function for a particular noise level σ

$$\ell(\theta; \sigma) = \mathbb{E}_{x \sim p_{\text{data}}(x), \tilde{x} \sim p_\sigma(\tilde{x}|x)}[||s_\theta(\tilde{x}, \sigma) - \nabla_{\tilde{x}} \log p_\sigma(\tilde{x}|x)||^2] \tag{28}$$

$$= \mathbb{E}_{x \sim p_{\text{data}}(x), \tilde{x} \sim p_\sigma(\tilde{x}|x)}[||s_\theta(\tilde{x}, \sigma) + \frac{\tilde{x} - x}{\sigma^2}||^2] \tag{29}$$

Combining this for all L noise levels, the overall objective is

$$\mathcal{L}(\theta; \{\sigma_i\}_{i=1}^L) = \frac{1}{L} \sum_{i=1}^{L} \lambda(\sigma_i) \ell(\theta; \sigma_i) \tag{30}$$

where $\lambda(\sigma_i)$ is a positive weighting coefficient which can be tailored to penalise the loss at specific noise levels. This is clearly a discretization of the SDE objective (11). Song and Ermon [2020] set $\lambda(\sigma) = \sigma^2$ to keep $\lambda(\sigma_i)\ell(\theta; \sigma_i)$ roughly the same order of magnitude for all $\{\sigma_i\}_{i=1}^L$. Using this weighting, the objective becomes

$$\mathcal{L}(\theta; \{\sigma_i\}_{i=1}^L) = \mathbb{E}_{l \sim \mathcal{U}\{1,L\}, x \sim p_{\text{data}}(x), \tilde{x} \sim p_{\sigma_l}(\tilde{x}|x)}[||\sigma_l s_\theta(\tilde{x}, \sigma_l) + \frac{\tilde{x} - x}{\sigma_l}||^2] \tag{31}$$

$$= \mathbb{E}_{l \sim \mathcal{U}\{1,L\}, x \sim p_{\text{data}}(x), \epsilon \sim \mathcal{N}(0,I)}[||\sigma_l s_\theta(x + \sigma_l \epsilon, \sigma_l) + \frac{x + \sigma_l \epsilon - x}{\sigma_l}||^2] \tag{32}$$

$$= \mathbb{E}_{l \sim \mathcal{U}\{1,L\}, x \sim p_{\text{data}}(x), \epsilon \sim \mathcal{N}(0,I)}[||\sigma_l s_\theta(x + \sigma_l \epsilon, \sigma_l) + \epsilon||^2] \tag{33}$$

Notably, this is equivalent to L_{simple} from the DDPM framework in (27) when we set $\epsilon_\theta(x_t, t) = -\sigma_l s_\theta(x_l, \sigma_l)$, revealing the underlying similarity between these two diffusion frameworks, and further connecting them under the umbrella of the overarching SDE framework.

Once the model has been trained and the optimal model parameters θ^* have been determined, given sufficient model capacity, $s_{\theta^*}(\tilde{x}, \sigma)$ will match $\nabla_{\tilde{x}} \log p_\sigma(\tilde{x}|x)$ almost everywhere [Song and Ermon, 2020]. Sampling can be conducting using a variety of methods that make use of these scores. One method that is proposed in Song and Ermon [2020] is annealed Langevin dynamics as shown in Algorithm 2, which is also

a potential 'corrector' to use in PC sampling from the SDE framework. Samples are initialised from the prior distribution $\mathcal{N}(0, \sigma_L^2 I)$ and then Langevin dynamics is run progressively over the decreasing noise level to sample from $p_{\sigma_i}(x)$ for each i until samples are from $p_{\sigma_1}(x) \approx p_{\text{data}}(x)$ when $\sigma_1 \approx 0$.

Since $\{p_{\sigma_i}\}_{i=1}^L$ are perturbed with Gaussian noise, their supports span the whole space, ensuring their scores are well defined. Also, assuming Langevin dynamics produces good samples for a perturbed distribution p_{σ_i}, the samples will likely come from high density regions of p_{σ_i}. This also means that the samples likely come from high density regions of $p_{\sigma_{i-1}}$, since $p_{\sigma_{i-1}} \approx p_{\sigma_i}$. Since score estimation performs better in high density regions, samples from p_{σ_i} are a good starting point for running Langevin dynamics on $p_{\sigma_{i-1}}$, so the samples can be passed between each perturbed distribution until samples from p_{σ_1} are produced.

Algorithm 2 Annealed Langevin dynamics used for sampling with SGM framework

Require: $\{\sigma_i\}_{i=1}^L$, ϵ, T
  Initialise $\tilde{x}_0 \sim \mathcal{N}(\tilde{x}_0; 0, \sigma_{\max}^2 I)$
  for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \frac{\sigma_i^2}{\sigma_L^2}$
    for $t \leftarrow 1$ to T **do**
      $z_t \sim \mathcal{N}(0, I)$
      $\tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{x}_{t-1}, \sigma_t) + \sqrt{\alpha_i} z_t$
    end for
    $\tilde{x}_0 \leftarrow \tilde{x}_T$
  end for
  return $\tilde{x}_0$

2 Conditional Simulation Techniques

The discussion in the previous chapter focused on various types of diffusion models and how they can generate unconditional samples from a target distribution. However, unconditional generation is only part of the puzzle when considering the success of deep generative models, particularly when looking at real world use cases. In many domains, the ability to produce high quality data with a given condition is a critical task. Conditional generation offers control over the samples produced so that they conform to the requirements of a specific application. Potential use cases are numerous, including image generation (image inpainting, super-resolution, text-to-image), protein design (motif-scaffolding problem), healthcare and finance (time-series imputation) as well as many others.

Diffusion models not only achieve state of the art performance in unconditional generation, but can also incorporate conditional information into their samples. Some examples of conditional generation applications are displayed in Figure 7, which shows an image inpainting technique called RePaint proposed in Lugmayr et al. [2022], based on DDPM, and Figure 8, which shows an image super-resolution approach named SR3 proposed in Saharia et al. [2021], also using a DDPM. These are examples of two models which aim to solve similar problems - conditional image generation based on a partial image input (a blurry image in the case of Saharia et al. [2021] or an obscured image in Lugmayr et al. [2022]). Despite the similarity of their objectives, however, SR3 and RePaint take very different approaches. SR3 trains a DDPM tailored for conditional generation by supplying conditional information at training time to the neural network. RePaint, however, uses a pretrained unconditional DDPM and injects the condition into the sample generation process, not requiring a specific conditional model to be trained. These two fundamentally different approaches for seemingly similar tasks highlight the wide array of methods available for incorporating conditional information into diffusion models.

In this chapter, we will explore the theory behind three different conditional simulation techniques. Models similar to that used by SR3 we will term tailored conditional models, which is the first technique we will discuss. We will also discuss two methods similar to that used in RePaint which only require an unconditional model and

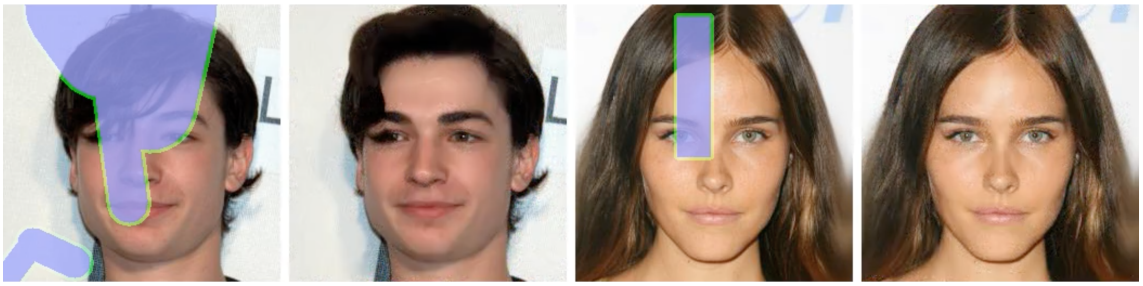


Figure 7: Image inpainting by RePaint in Lugmayr et al. [2022] with associated masked input.

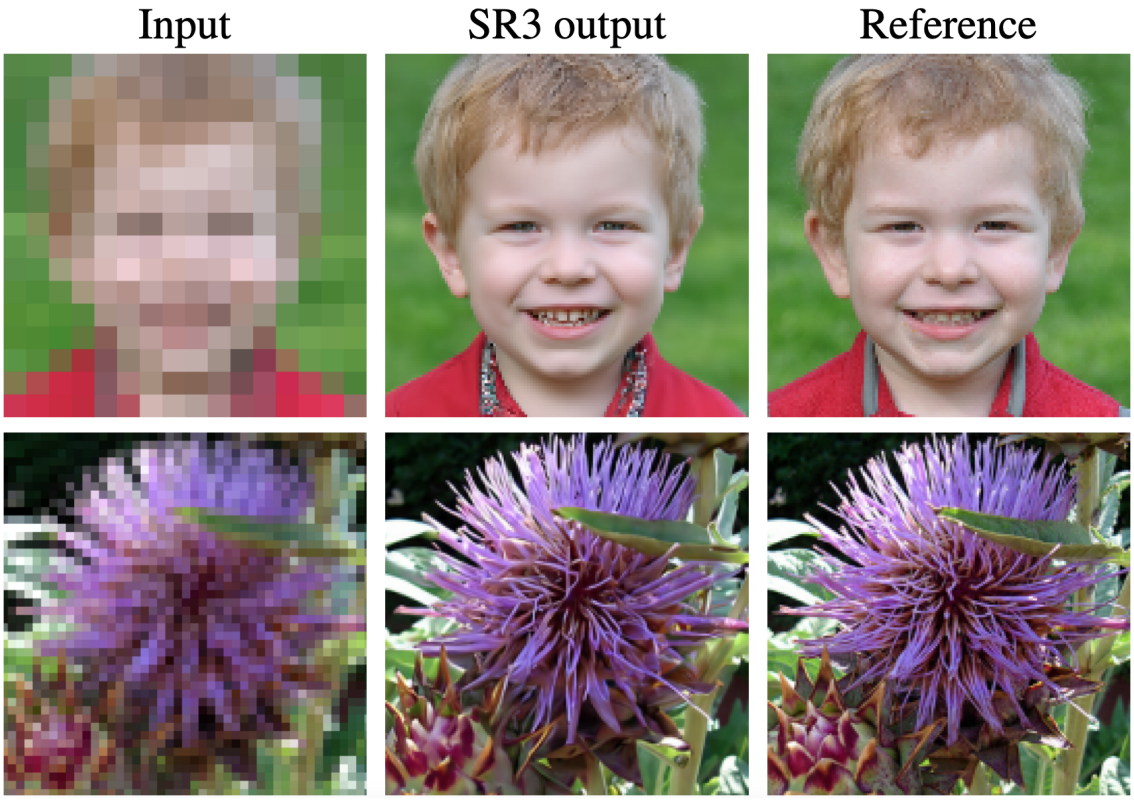


Figure 8: Super-resolution by SR3 from Saharia et al. [2021] with associated low resolution input and reference image.

enforce conditions during sample generation. The replacement method, proposed in Song et al. [2021] and named in Ho et al. [2022], is the more basic of the two, which forward diffuses the conditioning information (such as the unmasked part of the image in the RePaint examples) and injects the diffused condition into the intermediate sample at each timestep during generation. SMCDiff [Trippe et al., 2023] improves upon the replacement method by looking ahead one timestep and weighting each sample by how likely the future diffused condition is, using particle filtering to correct for error in the replacement method at the expense of more computation.

Since the replacement and SMCDiff methods require diffusing the conditioning information, we will be focusing on conditions that can be diffused, such as partial images used in inpainting. Conditions such as class labels are not directly applicable to the replacement and SMCDiff methods, and so will not be discussed in this chapter or the later experiments.

These three methods of conditioning can be applied to any of the previously discussed diffusion model frameworks. While the following discussion will focus mainly on the SDE framework since it is the most general and encompasses continuous versions of DDPM and SGM, the theory is broadly applicable to all three frameworks without much need for adaptation.

For unconditional generation, a diffusion model aims to generate samples from some distribution $p_{\text{data}}(x)$. Now consider some conditional information y. The goal of conditional generation is to sample from the conditional distribution $p_{\text{data}}(x|y)$. Recall that for the SDE framework, a neural network is trained to learn the score function $\nabla_x \log p_t(x)$ so that the reverse SDE in (6) can be simulated. To sample from $p_{\text{data}}(x|y)$, the score we want to estimate is $\nabla_x \log p_t(x|y)$ to simulate the conditional reverse SDE

$$d\mathbf{x} = [f(x,t) - g(t)^2 \nabla_x \log p_t(x|y)]dt + g(t)d\bar{\mathbf{w}} \tag{34}$$

Tailored conditional models, the replacement method, and SMCDiff aim to estimate this score function in different ways, which we will now discuss.

2.1 Tailored Conditional Models

Tailored conditional models can be seen in application in many recent publications [Saharia et al., 2021, Li et al., 2021, Tashiro et al., 2021, Shi et al., 2022] and they involve a simple extension of the intuition behind unconditional diffusion models. Tailored conditional models seek to learn $\nabla_x \log p_t(x|y)$ directly with a neural network, giving rise to the objective function

$$\mathbb{E}_{t\sim\mathcal{U}(0,T),(x(t),y)\sim p(x(t),y)}[\lambda(t)||s_\theta(x(t),y,t) - \nabla_{x(t)} \log p(x(t)|y)||^2] \tag{35}$$

However, since we cannot calculate $\nabla_{x(t)} \log p(x(t)|y)$ in general, we cannot train the model using this objective. Batzolis et al. [2021] prove that the minimizer of (35) is the same as the minimizer of

$$\mathbb{E}_{t\sim\mathcal{U}(0,T),(x(0),y)\sim p(x(0),y),x(t)\sim p(x(t)|x(0))}[\lambda(t)||s_\theta(x(t),y,t) - \nabla_{x(t)} \log p(x(t)|x(0))||^2] \tag{36}$$

which is the same form as (11), except that the neural network $s_\theta(x(t),y,t)$ has the conditioning information y provided to it. Given the equivalence of the minimizers of (35) and (36), Batzolis et al. [2021] then prove that the optimal $s_{\theta^*}(x(t),y,t)$ trained with (36) is a consistent estimator of $\nabla_{x(t)} \log p(x(t)|y)$, or formally

$$s_{\theta^*}(x(t),y,t) \xrightarrow{P} \nabla_{x(t)} \log p(x(t)|y) \tag{37}$$

as the number of Monte Carlo samples used to estimate (36) goes to infinity. Note, however, that this assumes that there exists a unique $\theta^* \in \Theta$, where Θ is the parameter space, such that $s_{\theta^*}(x(t),y,t) = \nabla_{x(t)} \log p(x(t)|y)$, so the neural network must have sufficient capacity for consistency to hold.

Once the neural network has learned $\nabla_{x(t)} \log p(x(t)|y)$, sampling from $p_{\text{data}}(x|y)$ can then be done exactly as it is for an unconditional SDE diffusion model. The neural network can be substituted into (34) to give the estimated reverse SDE

$$d\mathbf{x} = [f(x,t) - g(t)^2 s_\theta(x,y,t)]dt + g(t)d\bar{\mathbf{w}} \tag{38}$$

and then any of the sampling methods mentioned in Section 1.1.1 can be used.

2.2 Replacement Method

The replacement method was first suggested as a method for solving general inverse problems with SDE diffusion models in Song et al. [2021]. It approaches conditional generation in a fundamentally different way to the tailored conditional method, because it does not train a specific neural network to learn $\nabla_{x(t)} \log p(x(t)|y)$. Instead, the replacement method uses an unconditional model that is trained on combined training data $z = (x, y)$ to learn $\nabla_{z(t)} \log p(z(t))$. The conditional gradient $\nabla_{x(t)} \log p(x(t)|y(t))$ can be extracted from the unconditional gradient, as described in Batzolis et al. [2021]

$$\nabla_{x(t)} \log p(x(t)|y(t)) = \nabla_{x(t)} \log p(x(t), y(t)) = \nabla_{z(t)} \log p(z(t))[: n_x]$$

where n_x is the dimensionality of x. The gradient $\nabla_{x(t)} \log p(x(t)|y(t))$ can then be used to approximate $\nabla_{x(t)} \log p(x(t)|y)$, and approximate conditional sampling can be done. This approximation $\nabla_{x(t)} \log p(x(t)|y(t)) \approx \nabla_{x(t)} \log p(x(t)|y)$ introduces error into conditional generation not present in the tailored conditional method. While the tailored conditional gradient estimator is consistent, Trippe et al. [2023] prove that the replacement method error is irreducible as it stems from the forward process itself, meaning that it cannot be eliminated by increasing the capacity of the neural network.

In practice, the replacement method can be implemented as described in Algorithm 3. For each step of the reverse process $y(t)$ in $z(t) = (x(t), y(t))$ is replaced with exact samples from the forward process $y(t) \sim p(y(t)|y(0))$. These exact samples can easily be obtained by diffusing the given condition y using the same forward process as in training. This process is shown in action in Figure 9 when used in RePaint for image inpainting. The conditional information y is the unmasked parts of image on the top row. Firstly, a reverse timestep is taken by the unconditional model, producing (using our notation) $z(t-1)$ from $z(t)$ as seen on the bottom row. Then the relevant parts of the image are replaced with exact samples from the forward diffusion of the conditioning information $y(t-1)$, forming the modified sample $z(t-1)$. This is then passed back into the unconditional model, and the process is repeated until samples $z(0)$ are generated. The benefit of the replacement method compared to the tailored conditional method is clear when considering this RePaint example. The unconditional diffusion model could be any pretrained image generation diffusion model, and no extra training is required to condition samples. On the other hand, the tailored conditional method would have to train its own specific conditional model.

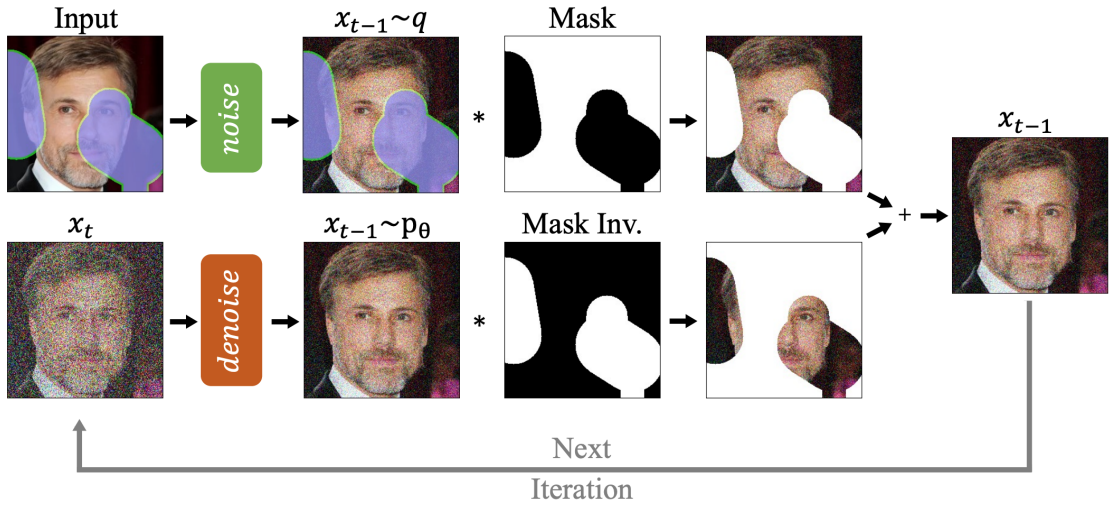


Figure 9: An overview of the replacement method used in Lugmayr et al. [2022] for image inpainting.

Algorithm 3 Conditioning with the replacement method

Require: Condition $y(0)$, # of discretized timesteps T

$y_{1:T} \sim p(y_{1:T}|y(0))$ ▷ Forward diffuse condition

$z(T) \sim p_{\text{ref}}(z(T))$

for $t = T, ..., 1$ **do**

$z(t) = (x(t), y(t))$ ▷ Replace condition with exact sample

$z(t-1) = p_\theta(z(t-1)|z(t))$ ▷ Generate next sample with chosen sampling method

end for

return $z(0)$

2.3 SMCDiff Method

The SMCDiff method [Trippe et al., 2023] is a recently proposed conditional simulation technique that builds upon the replacement method, correcting its error using sequential Monte Carlo (SMC) techniques. This method was originally designed for protein generation as an approach to the motif-scaffolding problem. The biochemical functions of proteins can often depend on small subsets of residues [Wang et al., 2022], which we call a motif. The motif-scaffolding problem revolves around generating stable protein scaffolds to support a target motif, and it can therefore be framed as a conditional generation problem, where the motif is the condition for the generation of a protein design. The SMCDiff method has no specific ties to the domain of protein generation, however, and it can be applied to generic diffusion model applications.

SMCDiff, defined explicitly in Algorithm 4, approaches the approximation of $p_{\text{ref}}(x|y)$ as an SMC problem that can be solved with standard particle filtering methods such as those described in Doucet et al. [2001]. SMC techniques build upon ordi-

nary Markov chain simulations that look to approximate a distribution with a set of samples, or particles, as they weight each particle based on their likelihood. A resampling step is also generally included to avoid weight collapse over the course of many steps. In the replacement method, we have access to the noised condition $\{y(t)\}_{t=0}^{T}$, but at each timestep t only $y(t)$ is used to guide the sample generation towards $p_{\text{ref}}(x|y)$. SMCDiff uses this extra information available to look ahead one timestep to the less noised condition $y(t-1)$ and weight the samples $z(t)$ by how likely they are to produce $y(t-1)$, i.e. samples are weighted by $p_\theta(y(t-1)|z(t))$. This likelihood can be calculated easily since $p_\theta(z(t-1)|z(t))$ factorizes across y and x for each t. By applying particle filtering methods with these weights at each timestep, Trippe et al. [2023] prove that an $x(0)$ sample returned by Algorithm 4 converges in distribution to the true conditional distribution $p_{\text{data}}(x|y)$ as the number of particles $K \to \infty$, given the neural network has sufficient capacity in the unconditional model.

With this proof in mind, SMCDiff is the first algorithm that can generate asymptotically exact conditional samples from an unconditional diffusion model. Arbitrarily accurate conditional samples can be generated with sufficient computing power, and the parameter K provides a tradeoff between accuracy and computation required. Also, it is worth noting that while samples produced by Algorithm 4 are approximately distributed as $p_{\text{data}}(x|y)$, they are not independent, as they depend on the same instance of the diffused condition $\{y(t)\}_{t=0}^{T}$. Therefore, to generate independent conditional samples, Algorithm 4 must be run multiple times. Note that this lack of independence is true as well for samples from the replacement method generated by Algorithm 3.

Algorithm 4 Conditioning with the SMCDiff method

Require: Condition $y(0)$, # of discretized timesteps T, # of particles K

$\{y(t)\}_{t=0}^{T} \sim p(y(t)|y(0))$ ▷ Forward diffuse condition

$\forall k,\ z^k(T) \sim p_{\text{ref}}(z(T))$

for $t = T, ..., 1$ **do**

$\forall k,\ z^k(t) = (x^k(t), y^k(t))$ ▷ Replace condition with exact sample

$\forall k,\ w^k(t) = p_\theta(y(t-1)|z^k(t))$

$\forall k,\ w^k(t) = w^k(t) / \sum_{i=1}^{K} w^i(t)$

▷ Resample with chosen resampling method

$z^{1:K}(t) \sim \text{Resample}(w^{1:K}(t), z^{1:K}(t))$

▷ Generate next sample with chosen sampling method

$\forall k,\ z^k(t-1) = p_\theta(z(t-1)|z^k(t))$

end for

return $x^{1:K}(0)$

3 Experiments

We will conduct a variety of experiments to empirically compare the three conditional generation methods discussed in Section 2. We will investigate performance across datasets of varying dimensionalities and using conditions of varying complexities, to get a better understanding of how each method performs in practice. Across these experiments, the accuracy of the learned conditional distribution and training and generation time will be the main points of comparison, although other domain specific factors will also be discussed.

For these experiments, we implement diffusion models using the SDE framework. Specifically, for the forward process we use the VE SDE described in (12), since this definition typically produces better samples than the VP SDE [Song et al., 2021]. We closely follow the implementation of the VE SDE in Song et al. [2021], defining $\sigma(t) = \sigma_{\min}\left(\frac{\sigma_{\max}}{\sigma_{\min}}\right)^t$ so that the forward SDE and perturbation kernel become

$$d\mathbf{x} = \sigma_{\min}\left(\frac{\sigma_{\max}}{\sigma_{\min}}\right)^t \sqrt{2\log\frac{\sigma_{\max}}{\sigma_{\min}}}\, d\mathbf{w} \tag{39}$$

$$p(x(t)|x(0)) = \mathcal{N}\left(x(t); x(0), \sigma_{\min}^2\left(\frac{\sigma_{\max}}{\sigma_{\min}}\right)^{2t} I\right) \tag{40}$$

This definition requires us to set values for $\sigma_{\min}$ and $\sigma_{\max}$. Since the VE SDE framework is a continuous generalisation of the SGM framework, we use principles from the SGM framework as guidance. When t is small we want $x(t)$ to be close to $x(0)$, and we need $\sigma_{\min}$ to be small enough to allow this since, for small t, $p(x(t)|x(0)) \approx \mathcal{N}(x(t); x(0), \sigma_{\min}^2 I)$. Therefore, we set $\sigma_{\min} = 0.01$, as is done in Song et al. [2021], since this is small on the scale of all datasets used for these experiments. We then follow Technique 1 in Song and Ermon [2020] to set $\sigma_{\max}$ to the maximum pairwise distance between training samples. This will be tailored for each experiment.

As discussed in Section 1.1.1, there are many methods to sample from p_{data} using the SDE framework once the model has been trained. For these experiments, we discretize the reverse SDE using the Euler-Maruyama method with 1000 discrete timesteps, initializing samples from the prior $\mathcal{N}(0, \sigma_{\max}^2 I)$. While this sampling procedure can be improved with PC sampling, we chose to favour sampling speed over accuracy, especially since we found that the difference in sample quality between a basic Euler-Maruyama sampler and a PC sampler was minimal for our experiments.

For SMCDiff, we used systematic resampling for the resampling step in Algorithm 4. While there are many potential resampling methods, such as multinomial and residual, systematic resampling has been shown to outperform in terms of sample quality and computational complexity [Hol et al., 2006]. Also, to reduce unnecessary

resampling, we follow the implementation of SMCDiff in Trippe et al. [2023] by only conducting the resampling step when the weights become sufficiently non-uniform.

3.1 2D Synthetic Examples

For our first experiment, we will use two-dimensional datasets as described in Baptista et al. [2023] and Shi et al. [2022]. Consider three example joint distributions $p_{\text{data}}(x, y)$ where $y \sim \mathcal{U}[-3, 3]$ for all three and $p_{\text{data}}(x|y)$ is defined as:

$$\text{Example 1: } X = \tanh(Y) + Z,\ Z \sim \Gamma(1, 0.3)$$

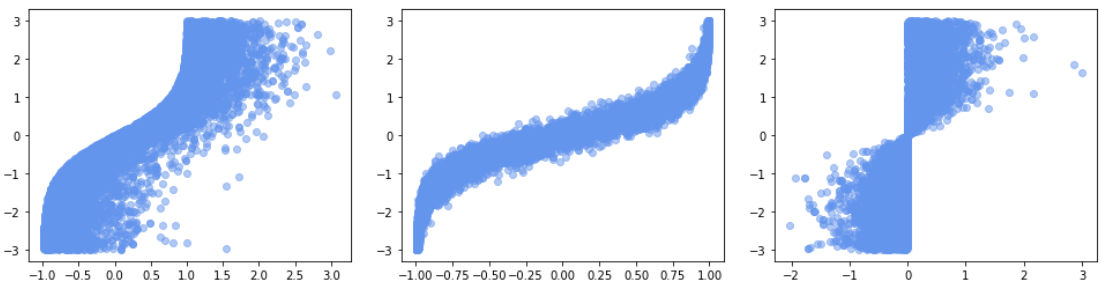
$$\text{Example 2: } X = \tanh(Y + Z),\ Z \sim \mathcal{N}(0, 0.05)$$

$$\text{Example 3: } X = Z \tanh(Y),\ Z \sim \Gamma(1, 0.3)$$

The shape of these distributions is shown in Figure 10. We will investigate how well each conditional simulation method can approximate the conditional density $p_{\text{data}}(x|y)$ for each example.

We set $\sigma_{\max} = 8$ as this was close to the maximum pairwise distance between training points for all three examples. For the replacement and SMCDiff methods we train an unconditional diffusion model that learns to generate (x, y) samples from the joint density $p_{\text{data}}(x, y)$. We do this as described in Section 1.1.1, using the objective function (11) with $\lambda(t) = 1$ to train a network $s_\theta(z(t), t)$ to learn $\nabla_{z(t)} \log p(z(t)|z(0))$, where $z = (x, y)$. For the tailored conditional method, we train a network $s_\theta(x(t), y, t)$ as described in Section 2.1 using (36), again with $\lambda(t) = 1$. We use the same architecture for $s_\theta(z(t), t)$ and $s_\theta(x(t), y, t)$ to ensure that model capacity is kept constant across all the simulation methods.

We use a similar simple neural network architecture to Shi et al. [2022] in their implementation of these 2D experiments. The network has four hidden layers and concatenates a sinusoidal embedding of the timestep t [Vaswani et al., 2023] to the output of the second layer. Overall, the model has just over $32,000$ parameters. For all three example datasets we train the models on $50,000$ training points drawn from the true distributions for 100 epochs, using a batch size of 64 and learning rate



(a) Example 1 (b) Example 2 (c) Example 3

Figure 10: 10,000 samples drawn from each of the example 2D distributions.

of 10^{-4}. On the CPU of a personal laptop the training time for both models was 210 seconds.

To assess each method we consider the conditions $y \in \{-1.2, 0, 1.2\}$ and draw 10,000 samples from the learned conditional distributions $p_{\text{data}}(x|y)$ of each method. For the tailored conditional method, 10,000 independent samples can be drawn in one run of a Euler-Maruyama sampler. However for the replacement and SMCDiff methods, as discussed in Section 2.3, samples from one run of their respective sampling algorithms are not independent, as they depend on a shared randomness in the forward diffusion of the condition y. To counteract this we conduct multiple smaller sampling runs using different instances of the diffused condition, to negate any bias induced by using a singular instance. Specifically, we conduct 10 sampling runs, drawing 1,000 samples each run, with 10 different instances of the diffused condition. For the SMCDiff method, we set the number of particles $K = 1000$.

Figure 11 shows the empirical densities produced by each conditional simulation technique with histograms, and the true conditional densities are overlaid with a solid line. We can see that the tailored conditional method gives the best approximation for all three examples, with the empirical densities matching the true densities very closely. The replacement method struggles noticeably in examples 1 and 2 to match the true densities accurately, but performs quite well in example 3. The SMCDiff method clearly improves upon the replacement method, as expected. In examples 1 and 2 the empirical densities more closely match the ground truths and they are only slightly worse than those produced by the tailored conditional method. In example 3, however, since the replacement method already learns the conditional density quite well, the resampling of the SMCDiff method does not yield any noticeable improvement.

To better quantify these results, Table 1 shows the mean of each learned $p_{\text{data}}(x|y)$ and compares it to the true means, with the closest estimate in bold. Again, the superior performance of the tailored conditional method is clear, as it produces the closest mean in all but one of the examples. This table further highlights how well this method learns the true distributions, as the empirical means all very close match the true means. Also shown is the time taken by each method to generate 10,000 samples. The tailored conditional method performs sampling the fastest, and SMCDiff is by far the slowest. It is worth noting, however, that some of the sample generation time for the replacement and SMCDiff methods can be attributed to the overhead of the sampler restarting with a different diffused condition, since we conduct their sampling over 10 runs. If we only ran the replacement and SMCDiff methods once to generate 10,000 samples, their sample generation times would be a more competitive 13 and 28 seconds respectively.

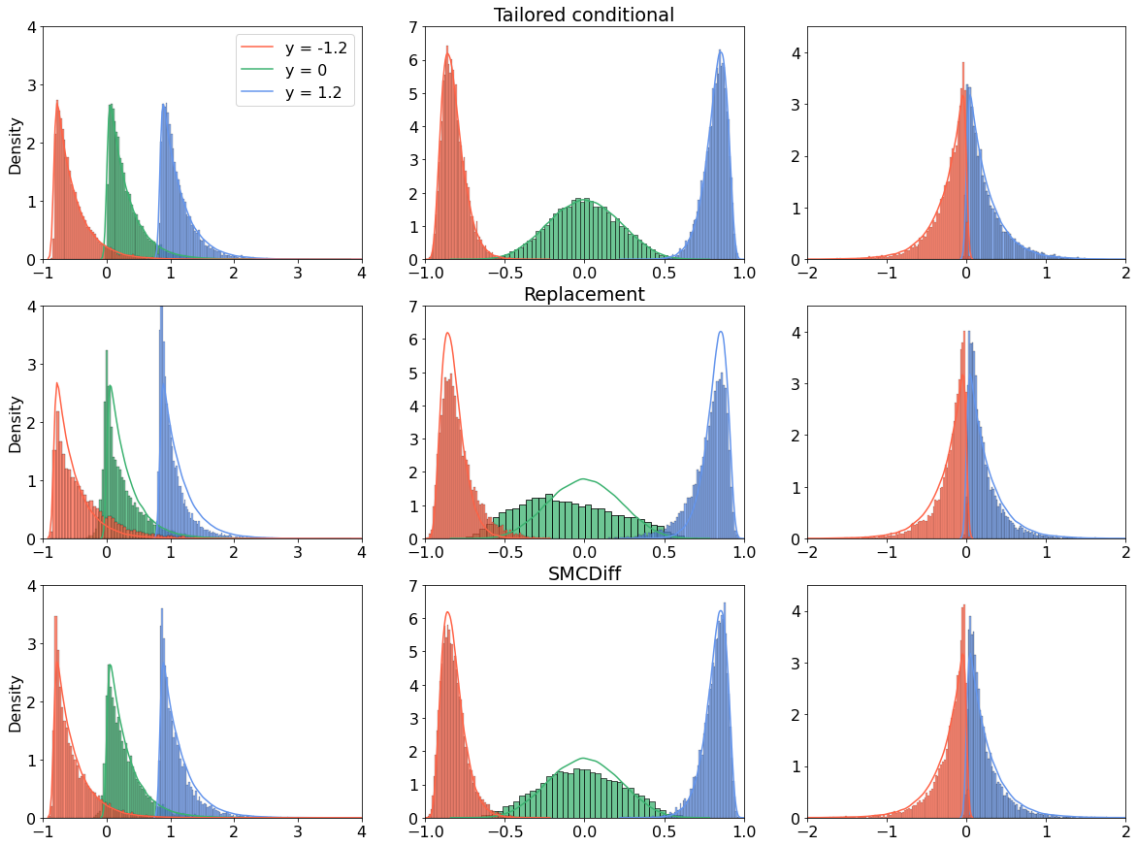


Figure 11: True posteriors $p_{\text{data}}(x|y)$ for 2D examples 1 (left), 2 (middle), and 3 (right) with $y \in \{-1.2, 0, 1.2\}$. Overlayed histograms are samples from the tailored conditional method (top), replacement method (middle), and SMCDiff method (bottom).

	Example 1			Example 2			Example 3		Time(s)
y_{obs}	-1.2	0	1.2	-1.2	0	1.2	-1.2	1.2	
Actual mean	-0.53	0.30	1.13	-0.82	0	0.82	-0.25	0.25	
Tailored conditional	-0.52	**0.31**	**1.12**	**-0.82**	**0**	**0.82**	**-0.24**	**0.25**	13
Replacement method	-0.38	0.25	1.03	-0.79	-0.11	0.78	-0.22	0.22	20
SMCDiff	**-0.53**	0.28	1.08	**-0.82**	-0.02	**0.82**	-0.22	0.22	40

Table 1: Means for conditional samples from $p_{\text{data}}(x|y)$ for 2D examples with $y \in \{-1.2, 0, 1.2\}$. Closest estimates to the true means are in bold. Rightmost column shows time for sample generation, to the nearest second.

3.2 Biochemical Oxygen Demand Bayesian Inference

We now consider applying conditional generation to a simple Bayesian inference problem. A model for biochemical oxygen demand (BOD) commonly used in water quality monitoring [Sullivan et al., 2010] is a popular test case for sampling methods [Marzouk et al., 2016, Shi et al., 2022]. The model describes the oxygen demand $Y(t)$ at time t, and can be defined by

$$Y(t) = A(1 - \exp(-Bt)) + \epsilon,\ \epsilon \sim \mathcal{N}(0, 10^{-3}) \tag{41}$$

which we further breakdown by

$$A = 0.8 + 0.4 \text{ erf}\left(\frac{X_1}{\sqrt{2}}\right),\ X_1 \sim \mathcal{N}(0,1) \tag{42}$$

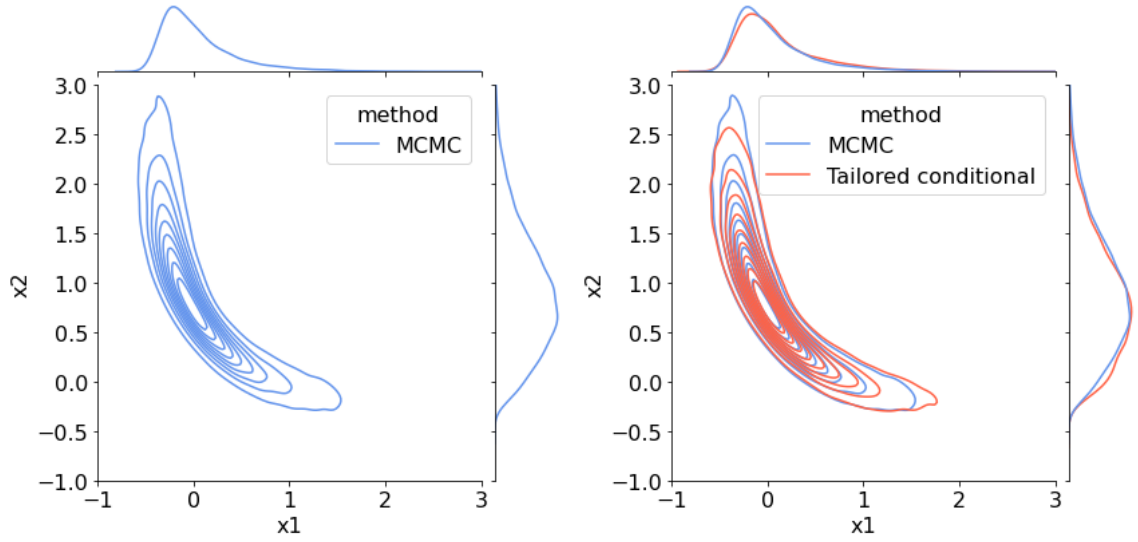
$$B = 0.16 + 0.15 \text{ erf}\left(\frac{X_2}{\sqrt{2}}\right),\ X_2 \sim \mathcal{N}(0,1) \tag{43}$$

For this example, we generate training data according to the above equations for times ranging from $t = 1$ to $t = 5$, and label $y = \{Y(t)\}_{t=1}^{5}$. Our training data, (x_1, x_2, y) is therefore seven-dimensional, and the target conditional density we seek to learn, $p_{\text{data}}(x_1, x_2|y)$, is two-dimensional.

The process for this experiment is much the same as the 2D experiment. We train neural networks with the same architecture as before with 50,000 training points. In this case, we train the models for 200 epochs as we found this led to better performance. Also, we found that conducting data scaling was necessary here. In our training dataset, the standard Gaussian variables x_1 and x_2 approximately ranged from -4 to 4, while the $\{Y(t)\}_{t=1}^{5}$ values were typically between 0 and 1. Given this difference in scales, before training we normalize each variable to have mean zero and unit variance. Once normalized, the maximum pairwise distance in the training data was close to 13, so we set $\sigma_{\max} = 13$. The training time for both neural networks, again on the CPU of a personal laptop, was 435 seconds.

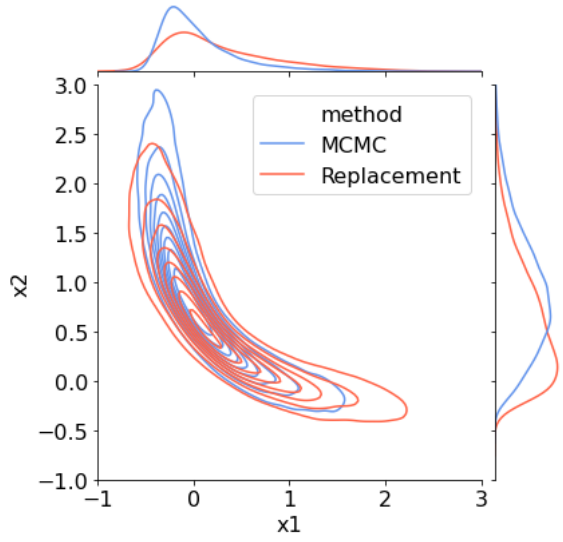
Considering the condition $y = [0.18, 0.32, 0.42, 0.49, 0.54]$, we draw 10,000 samples from the estimated densities $p_{\text{data}}(x_1, x_2|y)$ of each method. The sampling process is similar to the previous experiment, in that all 10,000 samples can be generated in one run for the tailored conditional method, but for the replacement and SMCDiff methods we conduct 10 separate runs with different instances of the diffused condition, producing 1,000 samples each run. For the SMCDiff method, the number of particles is set to $K = 1000$. Once the samples are generated, we cast them back to the original data scale. In this experiment, since the conditional density $p_{\text{data}}(x_1, x_2|y)$ is not readily available, we cannot generate ground truth reference samples by simply sampling from a tractable density. Marzouk et al. [2016] use an adaptive Metropolis-Hastings MCMC scheme to generate exact samples from the conditional density, running the scheme for 6×10^6 steps and discarding 2×10^4 as burn-in. To verify their generated data, and have reference points available for plotting, we run our our Metropolis-Hastings MCMC sampler for 10^6 steps and discard 4×10^4 as burn-in. The moment statistics of our MCMC samples agree with those stated in Marzouk et al. [2016].

Figure 12 shows the empirical posterior densities $p_{\text{data}}(x_1, x_2|y)$ for each conditional simulation method along with the true density derived from our MCMC samples.

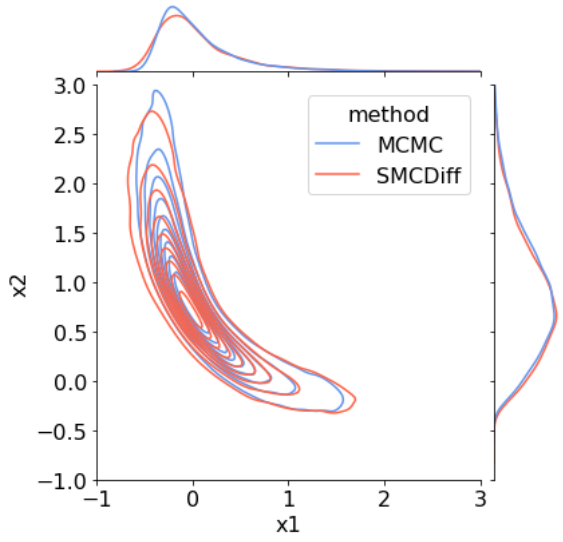


(a) MCMC samples

(b) Tailored conditional method



(c) Replacement method



(d) SMCDiff

Figure 12: Estimated posterior densities $p_{\text{data}}(x_1, x_2|y)$ with marginals from BOD example with $y = [0.18, 0.32, 0.42, 0.49, 0.54]$. True density from MCMC samples shown in blue.

The relative performance of each of the simulation methods is similar to the first experiment. The replacement method produces the least accurate posterior density, especially when looking at the x_2 marginal. Both the SMCDiff and tailored conditional methods perform much better, matching the true density closely. Visually, there is not much of a difference between the empirical densities for either of these methods, and there is no clear better performing method. This is different than in the first experiment, where the tailored conditional method was clearly the strongest.

	Mean		Variance		Skew		Kurtosis		Time(s)
	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2	
True	0.075	0.875	0.190	0.397	1.94	0.681	8.54	3.44	
Tailored conditional	0.108	**0.835**	**0.203**	0.363	1.83	0.707	**5.06**	0.566	17
Replacement method	0.284	0.616	0.373	0.373	1.22	0.945	1.63	**0.975**	20
SMCDiff	**0.085**	0.830	0.241	**0.389**	**1.85**	**0.691**	4.50	0.637	40

Table 2: Posterior moments of estimated $p_{\text{data}}(x_1, x_2|y)$ from BOD example with $y = [0.18, 0.32, 0.42, 0.49, 0.54]$. Closest estimates to reference MCMC statistics are in bold. Rightmost column shows time for sample generation, to the nearest second.

Table 2 shows moment statistics for the generated samples and compares them to the moments of the MCMC samples reported in Marzouk et al. [2016]. The relative performance of the models here is similar to what we observe in Figure 12, with the SMCDiff and tailored conditional methods outperforming the replacement method. Indeed, these statistics emphasise the close performance of these two methods, as neither is superior across all moments. All methods estimate the mean, variance, and skew relatively well, but struggle especially on the higher order kurtosis moment. Furthermore, this table highlights the increased difficulty of this task compared to the first experiment, as no method matches the mean of the conditional density nearly as closely as the tailored conditional method did in experiment 1. Finally, the order of sample generation times is the same for this experiment, with the same caveat that if the SMCDiff and replacement method samples were produced in one run, they would have generation times of 17 and 33 seconds respectively.

3.3 MNIST Inpainting

For our final experiment, we will consider conditional simulation in the domain of image generation. While there are many common conditional image generation tasks, such as inpainting, super-resolution, and text-to-image, we will just focus on image inpainting for this experiment. We focus on the MNIST dataset which contains greyscale 28×28 images of handwritten digits [Deng, 2012], with some examples shown in Figure 13. We use masked images where the right half is occluded for our condition, and look to generate feasible complete images based on this.

Given the increased complexity of this task and dimensionality of the data, we use a more powerful model architecture for both the unconditional and conditional neural networks. An architecture with well established success for image data is the U-Net [Ronneberger et al., 2015], which is used in both Ho et al. [2020] and Song et al. [2021] for their diffusion models. We use this architecture which consists of an encoding path of repeated 3×3 convolutions and a decoding path of repeated 3×3 transposed convolutions, as shown in Figure 14. Beginning with a 28×28 sample, at each step of the encoding path we apply a 3×3 convolution, add a sinusoidal embedding of the timestep t, conduct group normalisation [Wu and He, 2018], and apply the swish activation function [Ramachandran et al., 2017]. We conduct four

Figure 13: Samples from MNIST dataset.

of these encoding steps, giving rise to five feature map resolutions (28×28, 26×26, 12×12, 5×5, 2×2) with an increasing number of feature channels. For each decoding step, we concatenate the relevant feature map from the encoding path, apply a 3×3 transposed convolution, add the timestep embedding, conduct group normalisation, and apply the swish activation function. Overall, this architecture has 1.1 million parameters. We train the models on the entire MNIST training dataset of 60,000 images for 100 epochs, with a batch size of 32, learning rate of 10^{-4}, and $\sigma_{\max} = 25$. Given the increased size of these models, training is unfeasible on the same personal laptop used in the previous experiments. As such, training is done on a Google Colaboratory notebook with GPU enabled, taking 45 minutes for each model.

For assessment, we use 10 masked conditions, one for each digit from 0-9, from the MNIST test dataset. Unlike the previous experiments, we are unable to sample from the true conditional distribution, so evaluating the performance of each method becomes a harder task. We can visually judge some samples produced by each method and how well they reconstruct the given condition in Figure 15, with samples for all 10 conditions in Appendix A. For the tailored conditional method we produce 256 samples for each condition in one run, but for the replacement and SMCDiff methods we conduct 16 sampling runs with different instances of the diffused conditions, producing 16 samples each run. We display a random sample of 16 digits from each method's sample set. For the SMCDiff method, we set $K = 512$. We observe that the replacement method struggles to produce coherent samples, even for the relatively easy '1' condition. While some of its samples are legible, the majority do not closely resemble the true digit, especially for the more difficult conditions '4' and '8'. The tailored conditional method and SMCDiff method perform much better, especially for the digit '1' as all of their samples are reasonable. Their per-

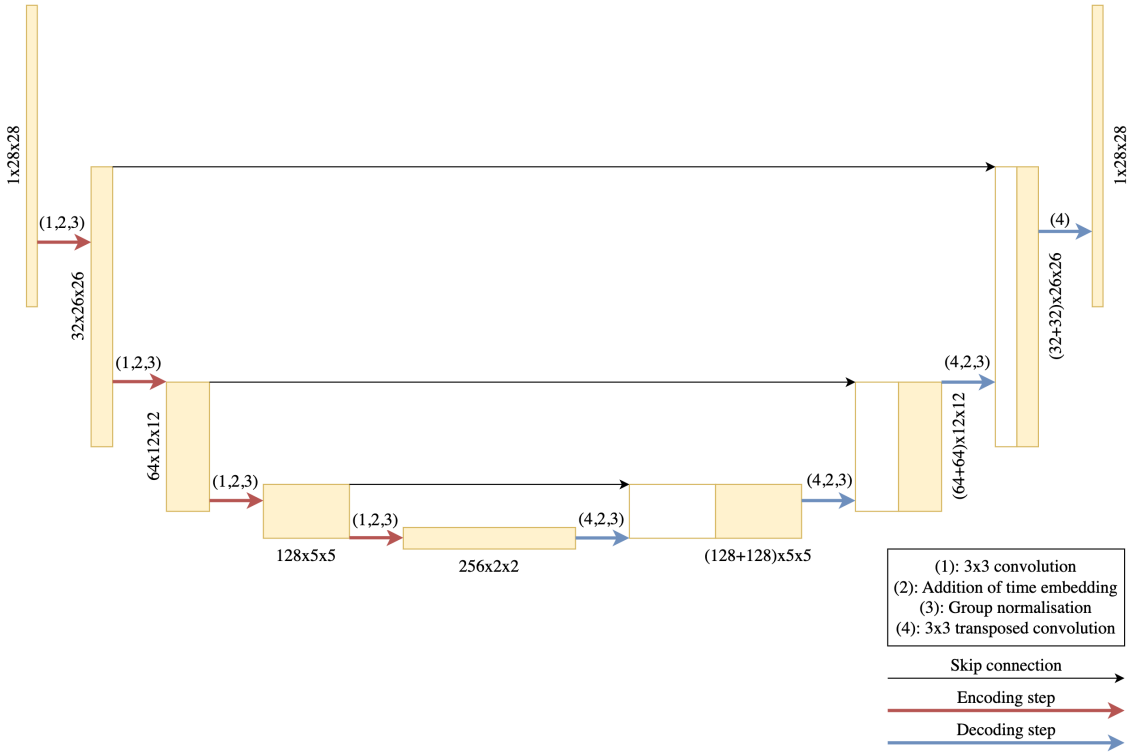


Figure 14: U-Net architecture. Swish activation function is applied to all steps following group normalisation, except the last.

formance degrades for the more complicated digits, and interestingly their relative performance is not consistent across digits. For the digit '4', the tailored conditional method appears to perform best, with the majority of its samples appearing quite close to the original image, while the SMCDiff method fails to produce many coherent digits. On the other hand, the opposite is true for the digit '8', as the tailored conditional method noticeably struggles in this case, while most of the SMCDiff samples are legible. Given this unstable performance, it is difficult to accurately rank the methods based on visuals alone. As such, we look to use quantitative metrics to assess how well generated samples reconstruct their ground truth images.

We use a few metrics which attempt to quantify the perceptual similarity between images, namely the peak signal-to-noise ratio (PSNR), structural similarity index (SSIM) [Wang et al., 2004], and learned perceptual image patch similarity (LPIPS) [Zhang et al., 2018]. PSNR is related to the mean-squared-error (MSE) between the predicted image P and true image T, where a higher score represents a better match. PSNR can be defined by

$$\mathrm{PSNR}(P,T) = 10\log_{10}\left(\frac{\max(P)^2}{\mathrm{MSE}(P,T)}\right) \tag{44}$$

SSIM correlates better with human judgement of similarity, since it looks to detect structural differences rather than the absolute error between images, in a manner

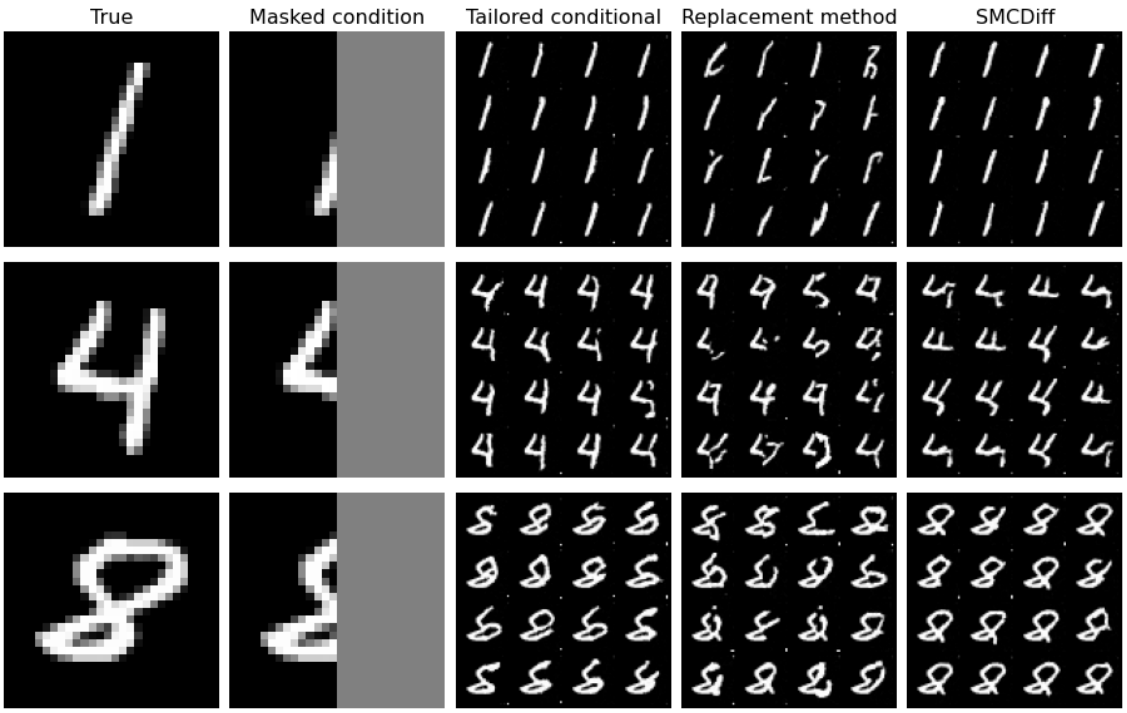


Figure 15: MNIST inpainting results. The true image from the MNIST test dataset is shown on the left, followed by the masked image supplied to the diffusion models as the condition. The rightmost three columns show 16 uncurated samples produced by each method for the given condition.

similar to human perception [Wang et al., 2004]. Again, a higher score represents a better match. SSIM compares local sections of pixels between images, and for a particular $n \times n$ window of an image it can be defined by

$$\text{SSIM}(P, T) = \frac{(2\mu_P\mu_T + c_1)(2\sigma_{PT} + c_2)}{(\mu_P^2 + \mu_T^2 + c_1)(\sigma_P^2 + \sigma_T^2 + c_2)} \tag{45}$$

where μ_i and σ_i are the mean and variance of the pixel values in the window of image i, σ_{PT} is the cross-correlation between the windows of P and T, and $c_1 = (0.01L)^2$, $c_2 = (0.03L)^2$, $L = 2^{\#\text{bits per pixel}} - 1$ are for numerical stability. SSIM is typically calculated using a sliding Gaussian kernel window of size 11×11 over the entire image, which is the procedure we use.

LPIPS is a more recently proposed metric which calculates the perceptual similarity between images by extracting features using a pre-trained deep neural network. It correlates better with human perception than both PSNR and SSIM [Zhang et al., 2018]. A low LPIPS score indicates perceptual similarity. In this experiment, we use features extracted from SqueezeNet [Iandola et al., 2016] which is a lightweight model which achieves similar performance to the successful deep network AlexNet [Krizhevsky et al., 2012] in image classification tasks with $50\times$ fewer parameters.

	PSNR (↑)	SSIM (↑)	LPIPS (↓)	Human preference (↑)	Time(s)
Tailored conditional	**15.7413**	**0.7712**	**0.0407**	44.5	260
Replacement method	13.1073	0.6124	0.0804	6.5	640
SMCDiff	15.6925	0.7626	0.0463	**49**	15680

Table 3: MNIST perceptual similarity metrics, as well as human judgement scores, for inpainting results for all 10 digit conditions. Numbers in bold indicate the best performance for the particular metric. Rightmost column shows time for sample generation, to the nearest second.

For each conditional simulation method, we now consider the entire sample sets generated previously. That is, we have 256 generated images for each of the 10 digit conditions, for a total of 2560 images for each method. We use the three discussed metrics to compare the perceptual similarity of the samples to their relevant ground truth image, and we report the results in Table 3. The best performing method for each metric is written in bold. Also, since all of these metrics attempt to emulate human judgement, we conduct a small scale experiment with three participants where, for a given condition, we present a sample from each of the three generation methods, and ask the participant to choose the sample that best represents the true image. We present 100 sets of samples to the participant in this manner, 10 for each digit, and display the average number of times each method was chosen as the best performing. There are of course shortcomings with this experiment, as this task is "cognitively penetrable", meaning that participants may choose to focus on certain arbitrary aspects of similarity, introducing subjectivity, especially considering our small sample size of participants [Zhang et al., 2018, Medin et al., 1993]. With this in mind, we will use all of the available metrics in tandem with these experiment results to assess model performance.

The tailored conditional method achieves the best score on all three evaluation metrics. The SMCDiff method follows closely in all metrics, while the replacement method achieves relatively poor results. For human judgement, the SMCDiff method performs best, closely followed by the tailored conditional method. This matches our intuitions from the samples in Figure 15, as the SMCDiff and tailored conditional methods perform almost equally as well, and certainly far better than the replacement method. We also report the time for sample generation in Table 3. More so than the previous experiments, the SMCDiff method is extremely slow in this case (over 60× slower than the tailored conditional method). This is because the number of particles K is higher than the actual number of samples we need. As mentioned previously, for the replacement and SMCDiff methods, we generate 16 samples for 16 separate diffusions for each digit. However, using $K = 512$, the SMCDiff method requires 512 particles at each step of the sampling process until sampling is complete, where we randomly select 16 of the 512 particles as the final sample. Therefore, 496 extra samples are essentially wasted per sampling run compared to the other methods, causing the extremely high generation time. This is an interesting flaw of the SMCDiff method, as when a high K is required to generate

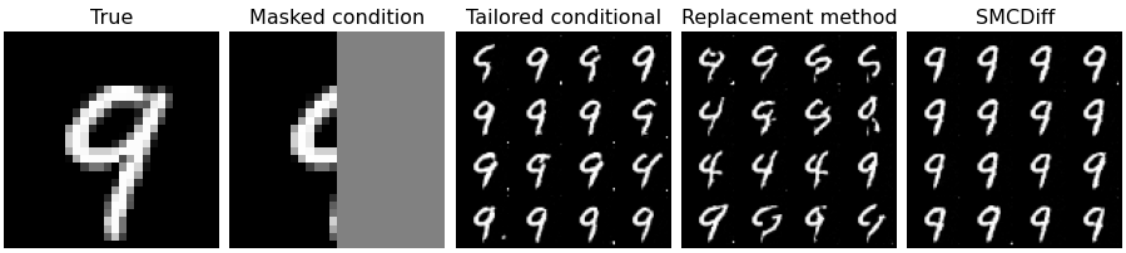


Figure 16: Inpainting results for the condition '9'.

accurate results but only a small number of samples is needed, there will be a lot of waste.

While the above evaluation judges how well each method reconstructs the true image from the condition y, perfect reconstruction should not be the goal of an image inpainting algorithm. Rather, we want to generate realistic images that are consistent with the condition y, but not necessarily identical to the true image. While metrics that assess the reconstruction error do correlate with model performance [Batzolis et al., 2021], they are too harsh in judging some samples. Observe, for example, the samples displayed in Figure 16, which are generated from a masked condition of the digit '9'. We can see that generated samples from the tailored conditional and replacement methods diverge into multiple digits, namely '9', '4', and '5'. While the samples that resemble '4' and '5' would be judged harshly by the above metrics, as they are not accurate reconstructions of the true image, they nevertheless respect the provided masked condition and are largely coherent digits. Therefore, we should incorporate some metrics which do not judge these as poor samples.

Essentially, we want to judge how the empirical distributions $p(x|y)$ agree with the true $p_{\text{data}}(x|y)$, as we do in experiments one and two. This is more challenging for this experiment, however, because we cannot easily obtain samples from the true conditional distribution. As a substitute for this, we can instead investigate how well generated samples agree with the unconditional MNIST disribution, using true samples from the MNIST test dataset. Fréchet Inception Distance (FID) [Heusel et al., 2018] is the metric most commonly used in image generation evaluation for this purpose, as it attempts to assess how close the distributions of two sets of images are. However, FID is not recommended for the MNIST dataset, since it was developed to work on larger colour images of natural scenery. As such, we instead conduct another small scale experiment, where participants are shown a sample alongside a set of true MNIST images and asked whether the sample belongs. We ask this for 50 samples from each method using multiple conditions, as well as 50 true MNIST samples which are separate from the reference set we show the participant. We present our results in Table 4. Interestingly, samples from the tailored conditional clearly perform much better than the SMCDiff method here, in contrast to the relatively even scores in Table 3. Judging on reconstruction error alone seems to unfairly penalises samples from the tailored conditional method, as it generates samples with more novelty than the SMCDiff method which seems to suffer from mode collapse, which can be

	% samples labelled accurate
True MNIST samples	80
Tailored conditional	58
Replacement method	22
SMCDiff	39

Table 4: Percentage of samples believed to belong to the MNIST dataset, as determined by a small scale human experiment with three participants.

seen in effect in Figure 16. The tailored conditional method better explores the multiple modes of the conditional distribution, producing reasonable samples which resemble '9', '5' and '4', while the SMCDiff samples collapse towards the same mode of the digit '9'.

4 Discussion

We will now go into a more in depth discussion of the results in Section 3 and their implications. We aim to explain why certain conditional simulation methods perform better in a given situation, and to establish which is the preferred method for a broad set of circumstances.

The most obvious takeaway from our experiments is the markedly poor performance of the replacement method. This is not unexpected, as this method introduces an irreducible error into the estimation of $p_{\text{data}}(x|y)$, as discussed in Section 2.2. Even still, we expected that a potential use case for the replacement method could arise when an unconditional diffusion model is readily available and training a separate conditional model would be prohibitively expensive, and the extra generation time of the SMCDiff method was unsuitable. However, the SMCDiff method so far outperforms the replacement method that such a scenario seems highly unlikely.

SMCDiff with a high number of particles K does incur substantial extra generation time since the complexity of Algorithm 4 is $\mathcal{O}(KT)$ where T is the number of timesteps. However, we observe that using small K, even with $K = 2$, still results in a large improvement in sample accuracy over the replacement method. Observe Figure 17 which shows this in effect for the MNIST inpainting experiment from Section 3.3. Calculating PSNR, SSIM, and LPIPS on samples generated by SMCDiff with $K \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$ (where $K = 1$ is equivalent to the replacement method) we see that all the metrics rapidly improve with K when small, before noticeably plateauing from about $K = 128$ onward. There is therefore great incentive to increase the number of particles when K is low with relatively little impact on the cost of generation. The increase in performance outweighs this time cost to such an extent that we suggest the replacement method is almost never favourable over the SMCDiff method with small K.

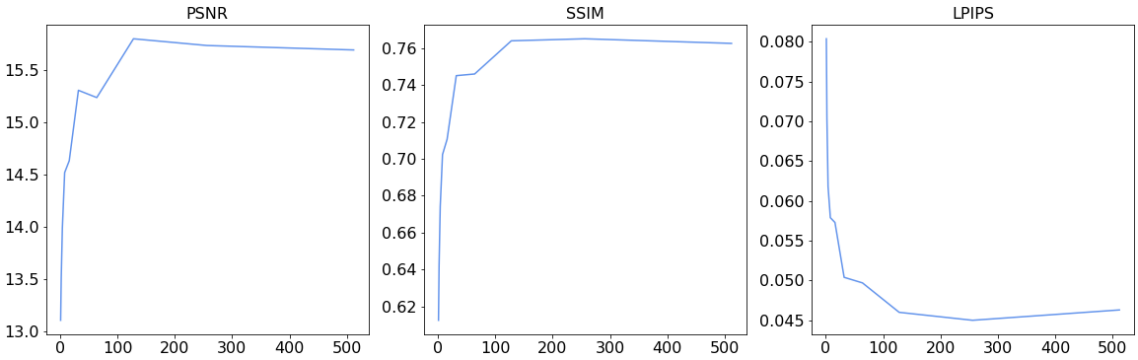


Figure 17: How the number of particles K used in SMCDiff for MNIST inpainting affects PSNR (↑), SSIM (↑), and LPIPS (↓). We calculate these values for $K \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$.

The comparison of the tailored conditional and SMCDiff methods is a more nuanced one, as there are valid use case scenarios for each. Prior to experimenting, we expected the tailored conditional method to always outperform SMCDiff in sample quality. This is because both methods use models with the same architecture, but the tailored conditional model can optimise its model capacity to incorporate conditional information into score estimation, while the SMCDiff model may have some of its capacity wasted on unneeded elements of unconditional score modelling.

More formally, since the tailored conditional method learns $\nabla_{x(t)} \log p(x(t)|y)$ directly, any error in its samples stem from the models approximation itself, which Batzolis et al. [2021] calls optimization error. On the other hand, the SMCDiff method approximates $\nabla_{x(t)} \log p(x(t)|y)$ with $\nabla_{x(t)} \log p(x(t)|y(t))$ and corrects for the error this induces with SMC methods. As such, error arises from both the models approximation of $\nabla_{x(t)} \log p(x(t)|y(t))$ and the SMC correction from $\nabla_{x(t)} \log p(x(t)|y(t))$ to $\nabla_{x(t)} \log p(x(t)|y)$, which Batzolis et al. [2021] calls approximation error. Given the same model capacity, assuming $\nabla_{x(t)} \log p(x(t)|y(t))$ and $\nabla_{x(t)} \log p(x(t)|y)$ have similarly difficult loss landscapes, the optimization error should be similar for both tasks, leading to a greater overall error for the SMCDiff method in general, since the approximation error ≥ 0.

From our experiments, however, we see that this expectation is not always the reality. In experiments two and three, samples from both methods are of very similar quality, with neither being clearly better. Despite the extra source of error for the SMCDiff method, it is able to perform on par with the tailored conditional method. This indicates that the loss landscape for $\nabla_{x(t)} \log p(x(t)|y(t))$ is likely easier to traverse than $\nabla_{x(t)} \log p(x(t)|y)$, in these two situations at least. Whether this is true in general, however, is not clear. Batzolis et al. [2021] suggest that learning $\nabla_{x(t)} \log p(x(t)|y(t))$ may be easier than $\nabla_{x(t)} \log p(x(t)|y)$ for the same reasons that $\nabla_{x(t)} \log p(x(t))$ is easier to learn than $\nabla_x \log p(x)$, as described in Section 1.1.3, but they do not provide any justification for this claim. Investigating when $\nabla_{x(t)} \log p(x(t)|y(t))$ is easier to learn than $\nabla_{x(t)} \log p(x(t)|y)$, or if it is true in general, is an area that requires future research, as this would provide guidance on whether the extra source of error for the SMCDiff method can be adequately offset by this easier optimisation.

Nevertheless, it is clear that the SMCDiff and tailored conditional methods can produce samples of similar quality. Therefore, when this is the case, the decision of which method to use depends largely on the time constraints of the particular scenario. If there is a suitable pretrained unconditional model available, which is not uncommon for popular use cases such as image and audio generation (e.g. the models from Song et al. [2021] and Rombach et al. [2022] produce high quality images and are readily available, as is the audio generation model from Kong et al. [2021]), then the SMCDiff method does not require any training at all, while the tailored conditional method requires a specific model to be trained. Furthermore, even if no unconditional model is readily available, its training is generally no harder than a specific conditional model, and that unconditional model can then generally

be used by the SMCDiff method with a broader scope of conditions than a tailored conditional model which generally is only suited for a specific use case. Consider, for example, an audio generation application, where possible conditional tasks include audio inpainting [Adler et al., 2012] and bandwidth extension. One tailored conditional model, unless carefully designed, would not be able to adequately perform both of these tasks, while the SMCDiff method could use the same unconditional base model for both. Therefore, if training time needs to be limited or multiple types of conditional generation tasks are required, the SMCDiff method is preferable to the tailored conditional method.

On the other hand, it is clear from our experiments that the tailored conditional method can perform sampling much faster than the SMCDiff method. In Sections 3.1 and 3.2, where we set K to be equal to the number of samples produced in each run, the SMCDiff method samples $2-3\times$ slower than the tailored conditional method. The difference in Section 3.3, where $K = 512$ but only 16 samples are produced each run, is more drastic because the number of particles is much higher than the number of samples, and the SMCDiff method is $60\times$ slower. Clearly, in applications where sampling speed is paramount, the tailored conditional method will be preferable in general. Furthermore, this preference will become stronger if only a small number of samples are generated per sampling run, since, assuming a high K value is needed for adequate sample quality, the difference in sampling times becomes starker. Overall, which conditional generation method to use, tailored conditional or SMCDiff, can be determined by considering whether training or generation time is more valuable. If it is unsuitable to devote time to training a potentially complex neural network, or multiple if various types of conditional information need to be considered, then this can be avoided with the SMCDiff method. However, if long sampling times are inappropriate, then the tailored conditional is preferable.

It is worth noting that the work we have done here is most relevant to inpainting-like tasks, as our three experiments are all general inpainting examples. There are of course many other types of conditional generation tasks, such as class conditioning, which our results may not be relevant for. Furthermore, we were unable to cover all available conditional generation methods in this dissertation. For example, two notable methods we did not cover are classifier guidance [Dhariwal and Nichol, 2021] and classifier-free guidance [Ho and Salimans, 2022] which both have the capacity to produce high quality conditional samples. Classifier guidance uses an auxiliary model, such as a classifier in the case of class conditioning, to augment the score estimate of an unconditional diffusion model during sampling based on the probability of the given condition arising, while classifier-free guidance mixes the scores of a tailored conditional model and an unconditional model, where the mixing weight determines the tradeoff between how well the condition is respected and the diversity of samples. As such, we must be clear that our recommendations are primarily for inpainting-like conditional generation, and they are not totally comprehensive. Given the relative novelty of diffusion models, new research is constantly being published, including for conditional generation techniques, and therefore this work will

not remain relevant unless it is consistently updated as new methods are established.

Bibliography

[1] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[2] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents, 2022.

[3] 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, Basile I M Wicky, Nikita Hanikel, Samuel J Pellock, Alexis Courbet, William Sheffler, Jue Wang, Preetham Venkatesh, Isaac Sappington, Susana Vázquez Torres, Anna Lauko, Valentin De Bortoli, Emile Mathieu, Sergey Ovchinnikov, Regina Barzilay, Tommi S Jaakkola, Frank DiMaio, Minkyung Baek, and David Baker. De novo design of protein structure and function with RFdiffusion. *Nature*, July 2023.

[4] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. RePaint: Inpainting using denoising diffusion probabilistic models, 2022.

[5] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement, 2021.

[6] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, nov 2022. doi: 10.1109/tpami.2021.3116668. URL https://doi.org/10.1109%2Ftpami.2021.3116668.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[9] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows, 2016.

[10] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[12] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

[13] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[14] Prafulla Dhariwal and Alex Nichol. Diffusion models beat GANs on image synthesis, 2021.

[15] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications, 2023.

[16] Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982. ISSN 0304-4149. doi: https://doi.org/10.1016/0304-4149(82)90051-5. URL https://www.sciencedirect.com/science/article/pii/0304414982900515.

[17] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. doi: 10.1162/NECO_a_00142.

[18] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation, 2019.

[19] Tianyu Pang, Kun Xu, Chongxuan Li, Yang Song, Stefano Ermon, and Jun Zhu. Efficient learning of generative models via finite-difference score matching, 2020.

[20] Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. Diffusion schrödinger bridge with applications to score-based generative modeling, 2023.

[21] Simo Särkkä and Arno Solin. *Applied Stochastic Differential Equations*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2019. doi: 10.1017/9781108186735.

[22] B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Universitext. Springer Berlin Heidelberg, 2010. ISBN 9783642143946. URL https://books.google.co.uk/books?id=EQZEAAAAQBAJ.

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

[24] G. Parisi. Correlation functions and computer simulations. *Nuclear Physics B*, 180(3):378–384, 1981. ISSN 0550-3213. doi: https://doi.org/10.1016/0550-3213(81)90056-0. URL https://www.sciencedirect.com/science/article/pii/0550321381900560.

[25] Ulf Grenander and Michael I. Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 56(4):549–603, 1994. ISSN 00359246. URL http://www.jstor.org/stable/2346184.

[26] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors. *Hand-*

book of Markov Chain Monte Carlo. Chapman and Hall/CRC, may 2011. doi: 10.1201/b10905. URL https://doi.org/10.1201%2Fb10905.

[27] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980. ISSN 0377-0427. doi: https://doi.org/10.1016/0771-050X(80)90013-3. URL https://www.sciencedirect.com/science/article/pii/0771050X80900133.

[28] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[29] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models, 2020.

[30] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. doi: 10.1162/089976603321780317.

[31] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models, 2022.

[32] 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, 2023.

[33] Haoying Li, Yifan Yang, Meng Chang, Huajun Feng, Zhihai Xu, Qi Li, and Yueting Chen. SRDiff: Single image super-resolution with diffusion probabilistic models, 2021.

[34] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional score-based diffusion models for probabilistic time series imputation, 2021.

[35] Yuyang Shi, Valentin De Bortoli, George Deligiannidis, and Arnaud Doucet. Conditional simulation using diffusion schrödinger bridges, 2022.

[36] Georgios Batzolis, Jan Stanczuk, Carola-Bibiane Schönlieb, and Christian Etmann. Conditional image generation with score-based diffusion models, 2021.

[37] Jue Wang, Sidney Lisanza, David Juergens, Doug Tischer, Joseph L. Watson, Karla M. Castro, Robert Ragotte, Amijai Saragovi, Lukas F. Milles, Minkyung Baek, Ivan Anishchenko, Wei Yang, Derrick R. Hicks, Marc Expòsit, Thomas Schlichthaerle, Jung-Ho Chun, Justas Dauparas, Nathaniel Bennett, Basile I. M. Wicky, Andrew Muenks, Frank DiMaio, Bruno Correia, Sergey Ovchinnikov, and David Baker. Scaffolding protein functional sites using deep learning. *Science*, 377(6604):387–394, 2022. doi: 10.1126/science.abn2100. URL https://www.science.org/doi/abs/10.1126/science.abn2100.

[38] A. Doucet, A. Smith, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Information Science and Statistics. Springer New York, 2001. ISBN 9780387951461. URL https://books.google.co.uk/books?id=uxX-koqKtMMC.

[39] Jeroen D. Hol, Thomas B. Schon, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 79–82, 2006. doi: 10.1109/NSSPW.2006.4378824.

[40] Ricardo Baptista, Bamdad Hosseini, Nikola B. Kovachki, and Youssef Marzouk. Conditional sampling with monotone GANs: from generative models to likelihood-free inference, 2023.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[42] Annett B. Sullivan, Dean M. Snyder, and Stewart A. Rounds. Controls on biochemical oxygen demand in the upper klamath river, oregon. *Chemical Geology*, 269(1):12–21, 2010. ISSN 0009-2541. doi: https://doi.org/10.1016/j.chemgeo.2009.08.007. URL https://www.sciencedirect.com/science/article/pii/S0009254109003404. Rates of Geochemical Processes and their Application to Natural Systems.

[43] Youssef Marzouk, Tarek Moselhy, Matthew Parno, and Alessio Spantini. Sampling via measure transport: An introduction. In *Handbook of Uncertainty Quantification*, pages 1–41. Springer International Publishing, 2016. doi: 10.1007/978-3-319-11259-6_23-1. URL https://doi.org/10.1007%2F978-3-319-11259-6_23-1.

[44] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[45] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[46] Yuxin Wu and Kaiming He. Group normalization, 2018.

[47] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[48] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.

[49] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

[50] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.,

2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[52] Douglas L. Medin, Robert L. Gladstone, and Dedre Genter. Respects for similarity, 1993.

[53] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

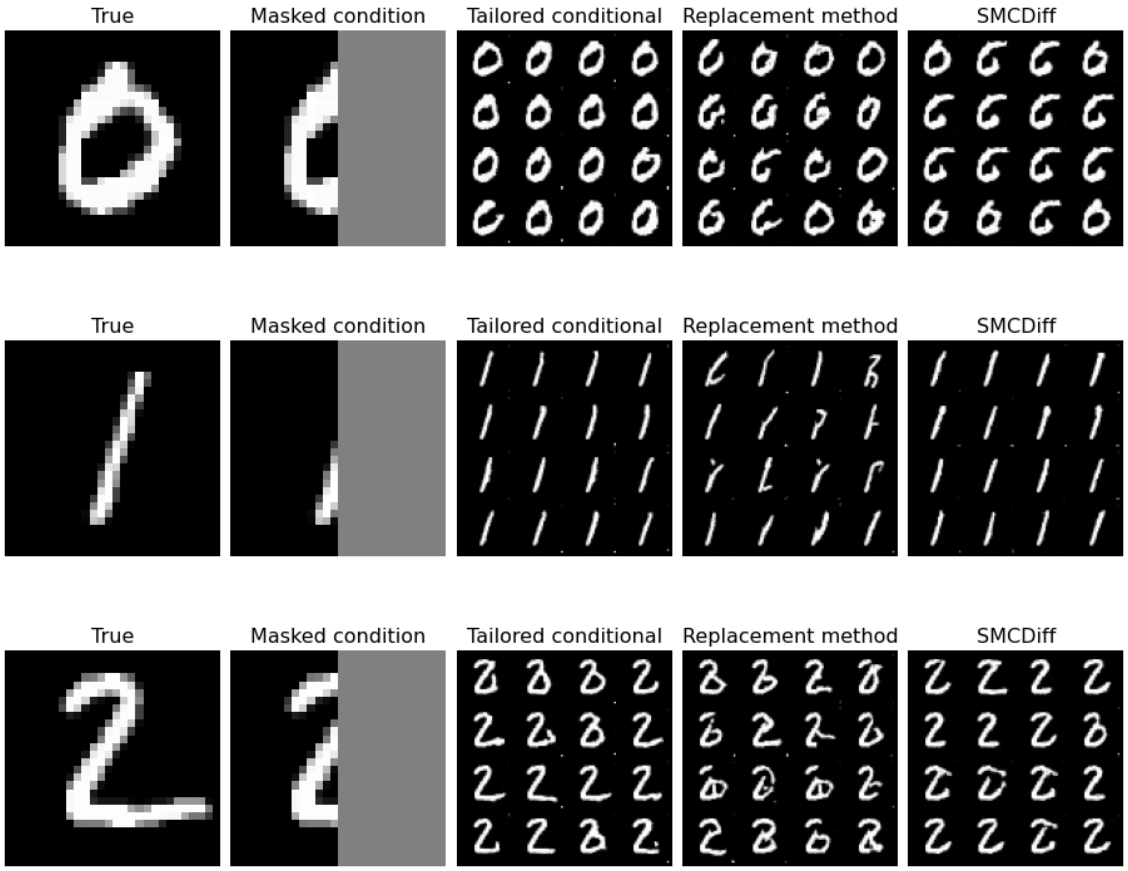
[54] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022.

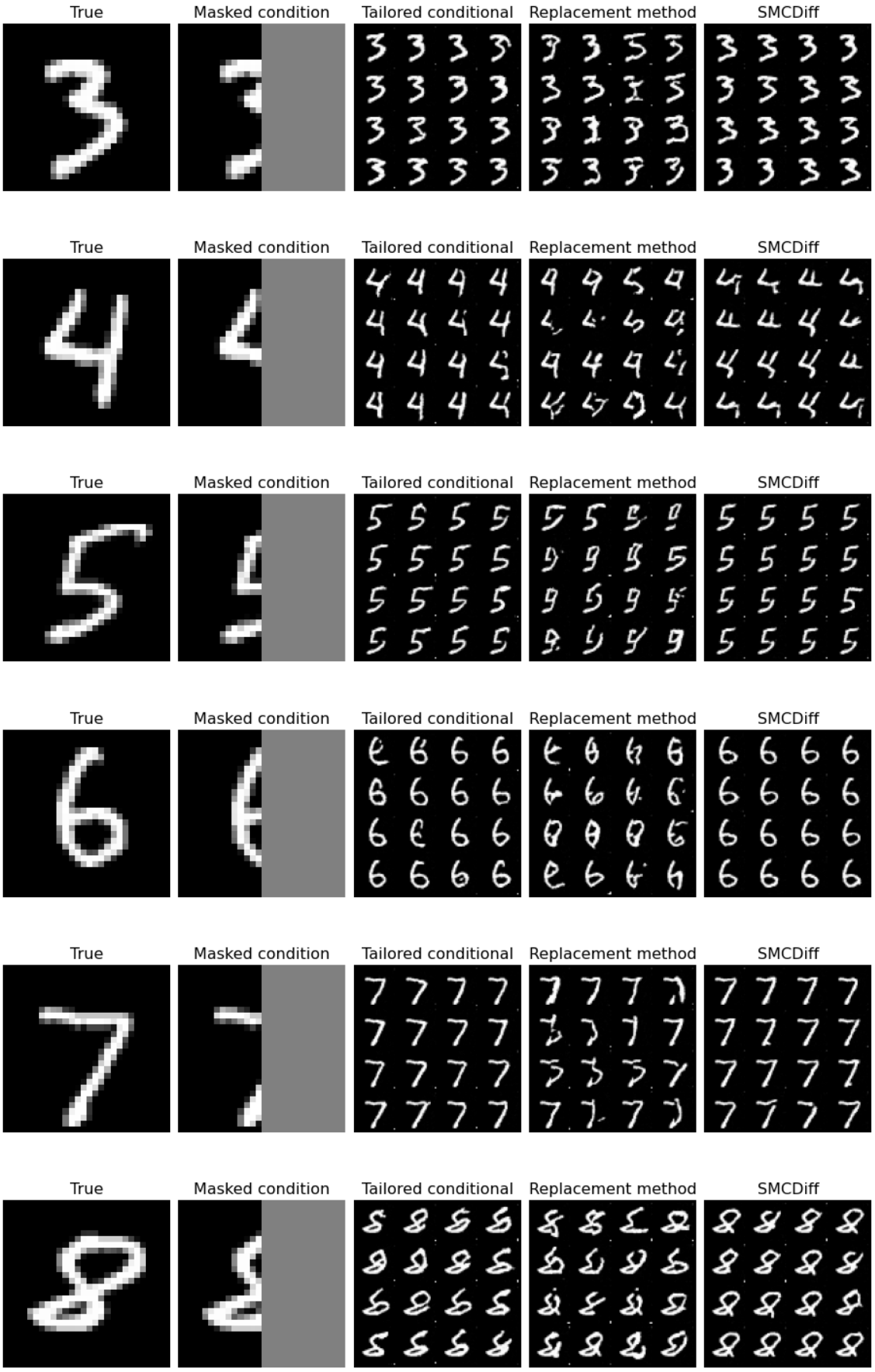
[55] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis, 2021.

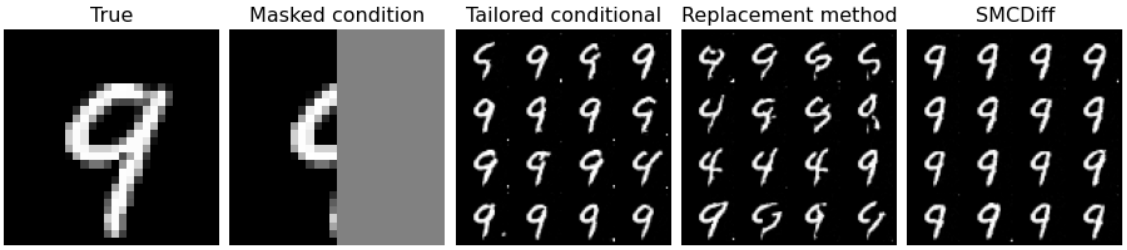
[56] Amir Adler, Valentin Emiya, Maria G. Jafari, Michael Elad, Rémi Gribonval, and Mark D. Plumbley. Audio inpainting. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(3):922–932, 2012. doi: 10.1109/TASL.2011.2168211.

[57] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

Appendix A


True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff


True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff
True
Masked condition
Tailored conditional
Replacement method
SMCDiff


True
Masked condition
Tailored conditional
Replacement method
SMCDiff