TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik und Informationstechnik
Self-Organizing Systems Lab

Analysis of Continuous-Time Diffusion Models for Discrete Data

Master-Thesis
Elektro- und Informationstechnik

Eingereicht von

Paul Leonardo Heller

am
17.04.2024

1. Gutachten: Prof. Dr. techn. Heinz Koeppl
2. Gutachten: Yannick Eich

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Paul Leonardo Heller, die vorliegende Master-Thesisgemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

English translation for information purposes only:

Thesis statement pursuant to §22 paragraph 7 of APB TU Darmstadt:

I herewith formally declare that I, Paul Leonardo Heller, have written the submitted thesis independently pursuant to §22 paragraph 7 of APB TU Darmstadt. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form. I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

Darmstadt, den 17.04.2024

_____

(Paul Leonardo Heller)

Abstract

In recent years, diffusion models have emerged as a promising alternative to traditional generative models by achieving state-of-the-art sample quality across various continuous data domains. However, research on their application to generate discrete data has been relatively limited. Moreover, previous attempts to extend diffusion models to discrete state spaces have focused on discrete-time frameworks, neglecting the inherent advantages and improved performance that continuous-time models have shown in continuous state spaces.

Recently, a work has filled this gap by adapting diffusion models to discrete state spaces within a continuous-time framework. This adaptation involves formulating the diffusion processes as continuous-time Markov chains and extending the Evidence Lower Bound to a continuous-time domain for training. Expanding on this work, another study has further developed the continuous-time Markov chain formulation of diffusion models by employing a novel score-based learning approach designed for general categorical data. Both approaches have shown promising initial results and are considered the current state-of-the-art for continuous-time diffusion models in discrete state spaces.

In light of these recent studies, this work analyzes their respective frameworks, known as τLDR and SDDM. In this analysis, we extend the investigation of these frameworks beyond their original scope by exploring a wider range of loss functions, parameterizations, and sampling methods. Through an empirical study, we compare these extensions to the original framework proposals and outline the strengths and limitations inherent in each framework. Our experiments are conducted on three different data domains: a low-dimensional binary spiral dataset, a mid-dimensional categorical maze dataset, and a high-dimensional image dataset, each presenting unique challenges and opportunities for yielding valuable insights. Our ultimate goal is to grasp the finer details of each modeling approach, enabling us to provide guidance on selecting the most suitable framework, its parameterization, loss function, and sampling strategy for generating different types of discrete data. Through this guidance, we intend to provide future researchers with a set of strategies for effectively modeling discrete data in continuous-time models, thereby laying the groundwork for advancing research in this field.

List of Abbreviations

Avg.	Average
BERT	Bidirectional Encoder Representations from Transformers
CIFAR-10	Canadian Institute for Advanced Research - 10
CNN	Convolutional Neural Network
CT-ELBO	Continuous-Time Evidence Lower Bound
CTMC	Continuous-Time Markov Chain
DTMC	Discrete-Time Markov Chain
D3PM	Structured Denoising Diffusion Models in Discrete State-Spaces - Framework in [3]
EBM	Energy-based model
ELBO	Evidence Lower Bound
FID	Frechet Inception Distance
FiLM	Feature-wise Linear Modulation
GELU	Gaussian Error Linear Unit
GPU	Graphics Processing Unit
IS	Inception Score
KL	Kullback-Leibler
MMD	Maximum Mean Discrepancy
MNIST	Modified National Institute of Standards and Technology
NFE	Number of function evaluations
ODE	Ordinary Differential Equation
PE	Positional Encoding
Prop.	Proportion
ResNet	Residual Neural Network
SDDM	Score-Based Continuous-Time Discrete Diffusion Models - Framework in [87]
SDE	Stochastic Differential Equation
τ**LDR**	Tau-Leaping Denoising Reversal - Framework in [7]

Notation

Numbers, Arrays, and Sets

θ	Parameters
a	A scalar
$\boldsymbol{a}$	A vector
A	A matrix
$\mathbb{1}$	A vector of ones
$\mathbb{I}$	Identity matrix
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^{H \times W}$	Set of real numbers over the dimensions H and W
$\mathbb{N}_0$	Set of natural numbers with 0 included
$\mathcal{T}, \mathcal{X}$	Calligraphic letters that denote a set
$\lvert\mathcal{X}\rvert$	Cardinality of a discrete set $\mathcal{X}$
$\{0, 1, \ldots, T\}$	Set of all integers between 0 and T

Indexing

x_t^d	The scalar entry of the d-th dimension at time t of vector $\boldsymbol{x}$
$\boldsymbol{x}^{\setminus d}$	All entries of $\boldsymbol{x}$ beside the d-th
$R_t(x, y)$	The (x, y) entry at time t of matrix R
$R_t^d(x, y)$	The (x, y) entry of the d-th dimension at time t of matrix R

Probability

$q(x), p(x), \pi(x)$	Probability distributions over a continuous or discrete variable x
X	A random variable representing outcomes of a random process
$\{X_t\}_{t \in \mathcal{T}}$	Collection of random variables X_t with time set $\mathcal{T}$, representing a random process
$\boldsymbol{X}$	A vector of random variables, representing multivariate outcomes
$\{\boldsymbol{X}_t\}_{t \in \mathcal{T}}$	Collection of vector random variables $\boldsymbol{X}_t$ with time set $\mathcal{T}$, representing a multivariate random process
$\mathbb{E}[\boldsymbol{x}]$	Mean of the vector $\boldsymbol{x}$
$\mathbb{E}[f(x)]$	Expectation of the function $f(x)$
$\mathbb{E}_{x \sim p}[f(x)]$	Expectation of the function $f(x)$ with respect to the distribution p
$\mathcal{D}_{\mathrm{KL}}(q(x) \parallel p(x))$	Kullback-Leibler divergence between $q(x)$ and $p(x)$
$\mathcal{U}(0, T)$	Uniform distribution with interval of $[0, T]$
$P(X_t = y)$	Probability of observing the value y at time t in the random process X

Operators and Calculus

A^T	Transpose of matrix A
$\mathrm{Tr}(A)$	Trace of matrix A
$\frac{\mathrm{d}}{\mathrm{d}x} f$	Derivative of f with respect to x
$\frac{\partial}{\partial x} f$	Partial derivative of f with respect to x
$\nabla_{\boldsymbol{x}} f$	Gradient of f with respect to $\boldsymbol{x}$

Functions

$f : \mathcal{X} \times \mathcal{T} \to \mathbb{R}$	A function mapping pairs from the Cartesian product of sets $\mathcal{X}$ and $\mathcal{T}$ to the set $\mathbb{R}$
$f^\theta(x, t)$	A parametrized function with inputs (x, t)
$\delta_{x,y}$	Kronecker delta for $x = y$
$\log(x)$	Natural logarithm of $x \in \mathbb{R}$
$\sin(x)$	Sine of $x \in \mathbb{R}$
$\cos(x)$	Cosine of $x \in \mathbb{R}$
$\exp(x)$	Exponential function of $x \in \mathbb{R}$
$\lfloor x \rceil$	Rounding function of $x \in \mathbb{R}$
$\mathrm{softmax}(\boldsymbol{x})$	Softmax function applied to vector $\boldsymbol{x}$

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Motivation and Related Works

Generative modeling is a central task in machine learning that aims to learn a probability distribution from data and generate new data from this learned distribution [3]. Among various techniques, diffusion models [83, 84, 36, 86] have become an important class of generative models. They achieve state-of-the-art performance in generating high-quality samples across various continuous data domains [51, 17], while admitting a stable, non-adversarial learning procedure [7]. Diffusion models were first introduced in [83] to model continuous and binary data distributions using concepts from non-equilibrium physics. Central to their design are two Markovian processes: a fixed forward corruption process and a learned reverse recovery process. The forward process gradually introduces noise into data samples until the data distribution is transformed into a simplified distribution [7]. Concurrently, the learnable reverse process seeks to denoise the corrupted data, effectively reversing the perturbations introduced along the forward process. This two-step approach enables data generation by first sampling noise from the simplified distribution and then gradually transforming it back through the learned reverse process into data samples approximately representing the original data distribution [83]. The reverse process is usually learned by using the Evidence lower bound (ELBO) [83, 36] or employing score matching techniques [41, 84].

While diffusion models have shown exceptional performance in generating continuous data distributions, research on modeling discrete data has been relatively limited. Some previous work [66, 64] has dealt with discrete data by embedding it into continuous space, upon which continuous space diffusion models are applied. Nonetheless, the intrinsic discrete nature of many data generation tasks — e.g., text, images, or biological sequences — makes embedding into continuous space not only non-intuitive but sometimes impractical [7].

Recognizing the limitations of previous methods, recent advances [38, 3] have adapted diffusion models to discrete state spaces. In [38], a multinomial extension of diffusion models in discrete state spaces was introduced, using uniform transition probabilities to perturb data during the forward process. This approach was further generalized in [3], introducing more structured data perturbation operations, including discretized Gaussian corruptions and absorbing state perturbations. Although these works have shown promising initial results, their studies are confined to a discrete-time framework. In such a framework, we can only learn data denoising at a finite number of predetermined time points in the process. This restriction limits the sampling scheme to basic ancestral sampling techniques [7, 87]. Conversely, in continuous-time frameworks, denoising can be learned at any arbitrary time point in the process, providing greater flexibility in defining the sampling scheme [7]. In continuous state spaces, this flexibility

has led to improved quality of the generated samples [86, 18] and reduced sampling times [43, 80]. Therefore, modeling discrete data in discrete state spaces within a continuous-time framework is also desirable. Nevertheless, the continuous-time frameworks in continuous-state spaces rely on stochastic differential equations (SDE) that involve estimations of the score function [41]. However, the score function is not defined for discrete random variables [40]. Consequently, these frameworks cannot be simply adapted to characterize a continuous-time extension for discrete diffusion [87].

To bridge this gap and leverage the advantages of continuous-time models for modeling discrete data directly in discrete state spaces, the τLDR framework [7] has recently been developed. This framework extends continuous-time diffusion models to discrete state spaces by reformulating the forward and reverse process as continuous-time Markov chains (CTMCs) [67, 78] and using a continuous-time version of the ELBO for training. Building on this work, the SDDM framework, presented in [87], uses the same CTMC formulation of the diffusion processes. However, instead of relying on ELBO, it employs a novel learning approach termed categorical ratio matching, which can be seen as an extension of score matching to general categorical data [87]. These frameworks have achieved promising results and superior performance over their discrete-time counterparts.

Given the success of these continuous-time models, additional continuous-time frameworks in discrete state spaces have been developed [4, 61]. In [4], the diffusion process was defined in probability simplex space within an SDE framework. However, their approach faces significant challenges regarding scalability for larger state spaces. On the other hand, in [61], a novel score entropy loss in discrete state spaces was developed. Nonetheless, their work is primarily designed for language modeling tasks.

Considering the generally limited research on continuous-time diffusion models in discrete state spaces and the proven benefits of the models in [7] and [87], we analyze their respective frameworks, τLDR and SDDM. In this analysis, we extend the initial scope of these frameworks by exploring a wider range of parameterizations, loss functions, and sampling methods. Through an empirical evaluation, we compare these extensions to the original framework proposals and outline the strengths and limitations inherent in each framework. Our empirical study spans three different discrete data domains, ranging from a low-dimensional binary spiral dataset through a mid-dimensional categorical maze dataset to a high-dimensional image dataset. Each dataset presents different challenges and opportunities for yielding profound insights. Our ultimate goal is to gain a deeper understanding of each modeling approach, allowing us to offer recommendations on the choice of framework, its parameterization, loss function, and sampling method for specific applications in discrete data modeling. By providing these recommendations, we aim to equip future researchers with a comprehensive basis and set of strategies for improving the generation of discrete data within continuous-time models.

1.2. Main Contributions

Our main contributions are:

- **Adaption of the continuous-time ELBO to the SDDM framework:** We adapt the continuous-time ELBO to the SDDM framework. This adaption broadens the range of applicable loss functions within SDDM, potentially improving the quality of the generated samples.

- **Generalization of sampling methods across frameworks:** We make sampling methods interchangeable between the τLDR and SDDM frameworks, enabling the use of a wider range of sampling methods than initially proposed.

- **Introduction of midpoint tau-leaping:** We introduce midpoint tau-leaping as a novel sampling strategy in the context of diffusion models to potentially improve sample quality.

- **Recommendations through experimental analysis:** We conduct experiments on three different discrete data domains. Based on our findings in these experiments, we guide in selecting the most suitable framework and sampling strategy for the specific discrete data domain. In addition, we offer suggestions for parameterizations and propose adjustments to loss functions to enhance performance.

- **Maze generation:** One of the experimental datasets involves the generation of mazes, a task not previously explored in generative modeling. This expands the scope of experimentation to novel data domains.

1.3. Structure

This thesis starts by providing a foundational overview of Markov chains in Chapter 2, introducing the required knowledge for diffusion models. Chapter 3 delves into the discrete-time formulation of diffusion models, highlighting their inherent limitations. In Chapter 4, we present the CTMC formulation of diffusion models and examine the distinct learning approaches in the τLDR and SDDM frameworks. Chapter 5 focuses on the architectural requirements for using the categorical ratio matching loss in SDDM. In Chapter 6, we describe and extend the methods employed in each framework to simulate the diffusion model processes efficiently. Chapter 7 outlines our experimental setup. In Chapter 8, we evaluate the different modeling approaches, and based on our findings, we provide recommendations for effectively generating different types of discrete data. Finally, in Chapter 9, we conclude and offer suggestions for future work to address identified limitations.

2. Background Markov Chains

This chapter provides the necessary background on Markov chains to understand the discussion on discrete-time diffusion models in Chapter 3 and to facilitate the derivation of diffusion models within a CTMC framework in Chapter 4. We begin by introducing discrete-time Markov chains (DTMCs), focusing on their essential properties for formulating diffusion models in discrete time. We then transition to continuous-time Markov chains, adapting these core properties to continuous time, thus setting the stage for the CTMC formulation of diffusion models.

2.1. Discrete-Time Markov Chains

Let $\mathcal{T} \subseteq \mathbb{N}_0$ be a time set and $\mathcal{X}$ be a discrete state space of finite cardinality $S = |\mathcal{X}|$, where we assume for simplicity that $\mathcal{X}$ is one-dimensional, making each $x \in \mathcal{X}$ a scalar value. Consider the t-indexed family of random variables $\{X_t\}_{t\in\mathcal{T}}$ to be a random process [77], with X_t taking values in $\mathcal{X}$. Then the random process $\{X_t\}_{t\in\mathcal{T}}$ is a discrete-time Markov chain if it satisfies the Markov property [67]

$$\begin{aligned} &P\left(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}, \ldots, X_0 = x_0\right) \\ &= P\left(X_{t+1} = x_{t+1} | X_t = x_t\right) = q_{t+1|t}(x_{t+1}|x_t) \end{aligned} \tag{2.1}$$

for all $t \in \mathcal{T}$ and for all $x_{t+1}, x_t, \ldots, x_0 \in \mathcal{X}$. Here, $q_{t+1|t}(x_{t+1}|x_t)$ denotes the one-step transition probability of being in state x_{t+1} at time step $t+1$, given that the state is x_t at time step t. The Markov property in Equation 2.1 states that the probability of transitioning from any state x_t at time step t to any other state x_{t+1} at time step $t+1$ is solely dependent on the current state x_t but not on the past states [67]. In other words, the future evolution of a DTMC is conditionally independent of the past, given the present.

When modeling experiments $\mathcal{T}$ is typically a finite set, e.g. $\mathcal{T} = \{0, 1, \ldots, T\}$. The random process $\{X_t\}_{t\in\mathcal{T}}$ is then fully described by its joint distribution

$$P\left(X_T = x_T, X_{T-1} = x_{T-1}, \ldots, X_0 = x_0\right) = q_{T:0}(x_{T:0}).$$

While the joint distribution of a general random process provides a complete description, it often lacks an intuitive or practical analysis structure [77]. In contrast, Markov chains simplify the analysis by allowing us to decompose the joint distribution. This decomposition can be achieved by iteratively applying the definition of the conditional distribution along with the

Markov property, which yields

$$
\begin{aligned}
&P\left(X_T = x_T, X_{T-1} = x_{T-1}, \ldots, X_0 = x_0\right) \\
&= P\left(X_T = x_T | X_{T-1} = x_{T-1}, \ldots, X_0 = x_0\right) \cdot P\left(X_{T-1} = x_{T-1}, \ldots, X_0 = x_0\right) \\
&= P\left(X_T = x_T | X_{T-1} = x_{T-1}\right) \cdot P\left(X_{T-1} = x_{T-1}, \ldots, X_0 = x_0\right) \\
&= P\left(X_T = x_T | X_{T-1} = x_{T-1}\right) \cdot P\left(X_{T-1} = x_{T-1} | X_{T-2} = x_{T-2}\right) \cdots P\left(X_0 = x_0\right) \\
&= P\left(X_0 = x_0\right) \cdot \prod_{t=1}^{T} P\left(X_t = x_t | X_{t-1} = x_{t-1}\right) = q_0(x_0) \prod_{t=1}^{T} q_{t|t-1}(x_t | x_{t-1}),
\end{aligned}
\tag{2.2}
$$

where q_0 is the initial state distribution at time $t = 0$. From Equation 2.2, we can see that a DTMC is fully defined by the one-step probabilities and the initial distribution.

2.1.1. Chapman-Kolmogorov Equation

The one-step transition probabilities only describe the probability of transitioning to a particular state in the next immediate step. By the Chapman-Kolmogorov equation [67], we can further describe the probabilistic evolution of a DTMC over multiple steps as follows

$$
\begin{aligned}
&P\left(X_{m+n} = y | X_0 = x\right) \\
&= \sum_{z \in \mathcal{X}} P\left(X_{m+n} = y, X_m = z | X_0 = x\right) \\
&= \sum_{z \in \mathcal{X}} P\left(X_{m+n} = y | X_m = z, X_0 = x\right) \cdot P\left(X_m = z | X_0 = x\right) \\
&= \sum_{z \in \mathcal{X}} P\left(X_{m+n} = y | X_m = z\right) \cdot P\left(X_m = z | X_0 = x\right),
\end{aligned}
\tag{2.3}
$$

where the sum is over all possible intermediate states $z \in \mathcal{X}$ at time step $m \in \mathcal{T}$ with $0 \leq m < n$. Equation 2.3 delineates that the probability of reaching a particular state $y \in \mathcal{X}$ in $m + n$ steps, starting from state x at time $t = 0$, is equal to the sum of the probabilities of each path to get there. In the next chapter, we will see that the computation of transition probabilities over multiple steps plays a crucial role in diffusion models, as it enables efficient training of such models [7].

2.1.2. Time-Reversal of a DTMC

For DTMCs, not only is the future conditionally independent of the past given the present, as seen in Equation 2.1, but the past is also conditionally independent of the future given the present [67]. Therefore, the Markov property

$$
P\left(X_{t+1} = x_{t+1} | X_t = x_t, X_{t-1} = x_{t-1}\right) = P\left(X_{t+1} = x_{t+1} | X_t = x_t\right)
$$

implies the following time-reversed statement

$$\begin{aligned} P(\underbrace{X_{t-1}=x_{t-1}}_{=:A} \mid \underbrace{X_t=x_t}_{=:B}, \underbrace{X_{t+1}=x_{t+1}}_{=:C}) &= \frac{P(A\cap B\cap C)}{P(B\cap C)} \\ &= \frac{P(C|A\cap B)\cdot P(A\cap B)}{P(C|B)\cdot P(B)} \\ &= \frac{P(C|B)\cdot P(A|B)}{P(C|B)} \\ &= P(A|B) \\ &= P(X_{t-1}=x_{t-1}|X_t=x_t)\,. \end{aligned} \tag{2.4}$$

This symmetric relation of the Markov property motivates looking at DTMCs in reversed time. It turns out that the time-reversed chain $\{\hat{X}_t\}_{t\in[0,T]} = \{X_{T-t}\}_{t\in[0,T]}$, is also a DTMC with one-step transition probabilities given by

$$\begin{aligned} &P\left(\hat{X}_{t+1}=x_{t+1}|\hat{X}_t=x_t,\hat{X}_{t-1}=x_{t-1},\ldots,\hat{X}_0=x_0\right) \\ &= P\left(X_{T-(t+1)}=x_{t+1}|X_{T-t}=x_t,X_{T-(t-1)}=x_{t-1},\ldots,X_T=x_0\right) \\ &= P\left(X_{T-(t+1)}=x_{t+1}|X_{T-t}=x_t\right) \\ &= P\left(\hat{X}_{t+1}=x_{t+1}|\hat{X}_t=x_t\right) \\ &= \frac{P\left(X_{T-(t+1)}=x_{t+1}\right)}{P\left(X_{T-t}=x_t\right)}\cdot P\left(X_{T-t}=x_t|X_{T-(t+1)}=x_{t+1}\right), \end{aligned} \tag{2.5}$$

where in the third line, we used the "reverse" Markov property from Equation 2.4 and in the fifth line, we applied Bayes' theorem to $P(X_{T-(t+1)}=x_{t+1}|X_{T-t}=x_t)$ [67]. This result has significant implications for learning the reverse process in diffusion models, where the goal is to reconstruct the original data distribution from a more noisy state. As we will see in the next chapter, this reconstruction can be achieved by modeling the reverse process as the time-reversed DTMC of the forward process, exploiting the symmetric nature of the Markov property [83].

2.2. Continuous-Time Markov Chains

Having previously explored DTMCs, we now extend our understanding to the continuous-time case. Let $\mathcal{T}\subseteq\mathbb{R}_{\geq 0}$ be a time set and consider $\{X_t\}_{t\in\mathcal{T}}$ as a random process, with X_t taking values in the discrete state space $\mathcal{X}$ of finite cardinality $S=|\mathcal{X}|$. Then $\{X_t\}_{t\in\mathcal{T}}$ is a CTMC if it satisfies the Markov property

$$P(X_t=y|X_r=x,\{X_v:0\leq v<r\}) = P(X_t=y|X_r=x) = q_{t|r}(y|x) \tag{2.6}$$

for all $r,t\in\mathcal{T}$, with $0\leq r<t$ and for all $x,y\in\mathcal{X}$ [67]. Here $q_{t|r}(y|x)$ represents the transition probability of being in state y at time t, given that the state is x at time r.

Figure 2.1 provides a schematic representation of a path of a one-dimensional CTMC with three states. Unlike in discrete-time processes, where state transitions occur only at discrete time steps, continuous-time processes allow transitions to other states at any moment. The continuous-time process repeatedly transitions from one state to another after having waited in the previous state for a randomly determined amount of time. Consequently, to satisfy the Markov property in Equation 2.6 and be independent of the past, a continuous-time process not only has to be independent of its past transitional states but also of the duration spent in the current state [67]. To meet this additional requirement, the time between two transitions has to be exponentially distributed, as the exponential random variable is the only continuous random variable with this memoryless property [67].

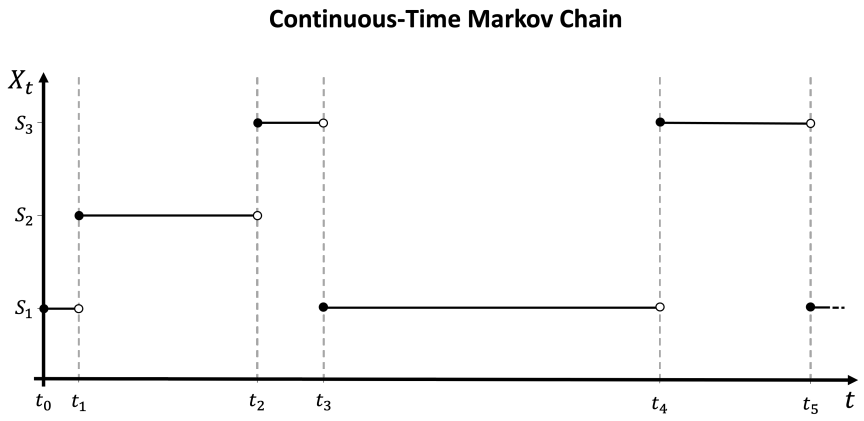


Figure 2.1.: Visualization of 1-D continuous-time Markov chain, based on [7].

2.2.1. Kolmogorov Equations for a CTMC

Analogous to DTMCs, CTMCs also satisfy the Chapman-Kolmogorov-Equation

$$\begin{aligned} P\left(X_{r+t} = y | X_0 = x\right) &= \sum_{z \in \mathcal{X}} P\left(X_{r+t} = y, X_r = z | X_0 = x\right) \\ &= \sum_{z \in \mathcal{X}} P\left(X_{r+t} = y | X_r = z, X_0 = x\right) \cdot P\left(X_r = z | X_0 = x\right) \\ &= \sum_{z \in \mathcal{X}} P\left(X_{r+t} = y | X_r = z\right) \cdot P\left(X_r = z | X_0 = x\right) \end{aligned} \tag{2.7}$$

for all $r, t \in \mathcal{T}$, where $0 \leq r < t$ and for all $x, y \in \mathcal{X}$. Equation 2.7 closely resembles its counterpart in discrete time, but in the continuous-time setting, we can derive a differential form of it. Such a derivation aligns more naturally with the continuous evolution of state probabilities over time and allows for a more granular analysis of the system's dynamics. By transitioning to differential equations, we can model the probability flows within the system at any infinitesimal moment, enhancing our capability to describe the system's behavior dynamically. These differential equations, known as the Kolmogorov forward and backward equations [50], lay the foundation for formulating diffusion processes as CTMCs [7]. Deriving these equations involves using an equivalent definition of CTMCs based on transition rate matrices $R_t \in \mathbb{R}^{S \times S}$

[67, 78]. A CTMC can then be completely described through the initial distribution q_0 and the transition rate matrix R_t [7]. The entries of the transition rate matrix can be understood as the transition intensities for infinitesimally small time intervals. In simpler terms, assuming the current state is x, then the next state y will likely be the one for which the transition matrix entry $R_t(x, y)$ is high, and further, the higher the rate is, the less time it will take for this transition occur [7]. Mathematically, the transition rate matrix R_t of a CTMC is defined as

$$R_t(x, y) := \lim_{h \to 0} \frac{P\left(X_{t+h} = y | X_t = x\right) - \delta_{y,x}}{h},$$

where $R_t(x, y)$ is the (x, y) element of the transition rate matrix R_t, h is an infinitesimally small time interval, δ is the Kronecker delta, and $P\left(X_{t+h} = y | X_t = x\right)$ is the infinitesimal transition probability that the process will be in state y at time $t + h$, given that the process is in state x at time t [67]. Conversely, the infinitesimal transition probabilities of a CTMC itself can be defined by the transition rate matrix

$$P\left(X_{t+h} = y | X_t = x\right) = \delta_{y,x} + R_t(x, y)h + o(h),$$

using the Landau notation such that $f(h) \in o(g(h))$ if $\lim_{h \to 0} \frac{f(h)}{g(h)} = 0$ holds [7]. From the transition rate matrix, we can further infer the following key properties [67]:

(i) $R_t(x, y) \geq 0, \forall x, y \in \mathcal{X}, y \neq x, \forall t \in \mathcal{T}$, since the transition rates refer to the probabilities of transitions.

(ii) $\sum_{y \in \mathcal{X}} R_t(x, y) = 0, \forall x \in \mathcal{X}, \forall t \in \mathcal{T}$, reflecting the principle that the total rate of leaving and staying in a state sums to zero.

(iii) $R_t(x, x) = -\sum_{y \neq x} R_t(x, y) \leq 0, \forall x \in \mathcal{X}, \forall t \in \mathcal{T}$, adhering to the convention that the rate of remaining in the same state is negative and equal to the negation of the sum of the rates of transitioning to other states.

With the transition rate matrix, we can derive a differential form of the Kolmogorov equation and fully describe the continuous probabilistic evolution of a CTMC [67]:

$$\text{Kolmogorov forward equation:} \qquad \frac{\partial}{\partial t} q_{t|r}(y|x) = \sum_{z \in \mathcal{X}} q_{t|r}(z|x) R_t(z, y)$$

$$\text{Kolmogorov backward equation:} \qquad \frac{\partial}{\partial r} q_{t|r}(y|x) = -\sum_{z \in \mathcal{X}} R_r(x, z) q_{t|r}(y|z).$$

In addition, the Kolmogorov forward equation also provides a differential equation for the marginals $P\left(X_t = x\right) = q_t(x)$ of the CTMC, described by

$$\frac{\partial}{\partial t} q_t(x) = \sum_{z \in \mathcal{X}} q_t(z) R_t(z, x). \tag{2.8}$$

The differential equation for the marginals in Equation 2.8 will be needed to determine convergence properties of the forward process in the CTMC formulation of diffusion models [7].

2.2.2. Time-Reversal of a CTMC

Just as we have established that the time-reversed chain is also Markovian for a DTMC in Equation 2.5, a similar principle applies to CTMCs. Consider a CTMC $\{X_t\}_{t\in\mathcal{T}}$ with a continuous time set $\mathcal{T} = [0, T]$, taking values in the discrete state space $\mathcal{X}$. Then the time-reversed process $\{\hat{X}_t\}_{t\in\mathcal{T}} = \{X_{T-t}\}_{t\in\mathcal{T}}$ is also a CTMC, whose transition probabilities $q_{r|t}(\cdot|\cdot)$ are given by (see proof in Appendix A.1 [87])

$$q_{r|t}(x|y) = \frac{q_r(x)}{q_t(y)} q_{t|r}(y|x), \qquad r < t.$$

Since we are considering a finite discrete state space $\mathcal{X}$, the reverse time process $\{\hat{X}_t\}_{t\in\mathcal{T}}$ is uniquely determined by its transition rate matrix $\hat{R}_t \in \mathbb{R}^{S\times S}$ [67], which satisfies (see proof in Appendix A.2 [7])

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

As we will see in Chapter 4, this fundamental characteristic allows us to model the continuous-time reverse diffusion process as the time-reversed CTMC, defined by $\hat{R}_t$, of the forward process [7, 87].

3. Background Diffusion Models

This chapter introduces the fundamental concepts of diffusion models required for this thesis and provides additional details on our motivation. In particular, we first present diffusion models in discrete time and discrete state spaces, as proposed in the D3PM framework [3]. Subsequently, we highlight the inherent limitations of these discrete-time models, the challenges they face, and the motivation for moving to a continuous-time approach. We then briefly present a continuous-time extension of diffusion models in continuous state spaces [86] and emphasize its compatibility issues for discrete state spaces. This discussion motivates the formulation of diffusion models in discrete state spaces within a CTMC framework.

3.1. General Setup for Discrete Diffusion Models in Discrete Time

The core concept behind diffusion models for generating data is to gradually introduce noise into clean data samples during the forward process until the resulting noisy data follows a known distribution, such as Gaussian or uniform [36, 3]. Subsequently, we employ a neural network to learn a reverse process to remove these noise perturbations. This approach allows the generation of new data samples by first sampling noise from the known distribution, which can then be transformed into data samples whose distribution closely approximates the original data distribution [7].

For simplicity, we initially consider generating discrete scalar data x_0 belonging to a discrete state space $\mathcal{X}$ with finite cardinality $S = |\mathcal{X}|$. We assume $x_0 \sim \pi_{\text{data}}(x_0)$ for some unknown discrete data distribution $\pi_{\text{data}}(x_0)$ [7]. In the discrete-time setting, we model the forward process as a DTMC $\{X_t\}_{t\in[0,T]}$, with X_t taking values in the discrete state space $\mathcal{X}$. During this process, the data samples x_0 undergo a series of transformations, defined by transition or corruption kernels $q_{t+1|t}(x_{t+1}|x_t)$. These transformations corrupt x_0 into a sequence of increasingly noisy latent variables $x_{0:T}$ with the same dimensionality as x_0. Mathematically, the forward process can be expressed as a joint distribution, which can be decomposed due to the Markov property, similar to Equation 2.2:

$$q_{0:T}(x_{0:T}) = \pi_{\text{data}}(x_0) \prod_{t=0}^{T-1} q_{t+1|t}(x_{t+1}|x_t). \tag{3.1}$$

After T time steps, this process yields a marginal distribution $x_T \sim q_T(x_T)$ that closely approximates an easy to sample reference distribution $p_{\text{ref}}(x_T)$.

In the reverse process, our goal is to sample x_T from the prior $q_T(x_T)$ and learn to transform it back to $x_0 \sim \pi_{\text{data}}(x_0)$. Although computing $q_T(x_T)$ is generally intractable, the convergence

of our forward process to the known reference distribution $p_{\text{ref}}(x_T)$ allows us to approximately sample x_T from $p_{\text{ref}}(x_T) \approx q_T(x_T)$ [83]. However, since this sampled x_T is completely noisy with almost no structure left, the discrepancy between x_0 and x_T is usually very high [98]. Consequently, when attempting to learn the entire transformation from x_T to x_0 at once in the reverse process, such a large discrepancy can lead to inefficiencies and slow convergence of optimization [11]. Instead, we aim to learn only the small perturbations between two consecutive latent variables x_{t+1}, x_t introduced by $q_{t+1|t}$ along the forward process. Thus, we effectively learn to gradually remove the introduced noise, which is more tractable than learning the entire transformation from x_T to x_0 at once [83].

If the amount of noise added at each step is sufficiently small, the forward and reverse process converges to the same functional form [21, 83]. This convergence allows the employment of a reverse process $\{\hat{X}_t\}_{t\in[0,T]}$, which is the time-reversed DTMC of the forward process. The joint distribution of the reverse process can then be inferred from $q_{0:T}(x_{0:T})$ via Bayes' rule [36]:

$$q_{0:T}(x_{0:T}) = q_T(x_T) \prod_{t=0}^{T-1} q_{t|t+1}(x_t \mid x_{t+1}), \qquad q_{t|t+1}(x_t|x_{t+1}) = \frac{q_{t+1|t}(x_{t+1}|x_t) q_t(x_t)}{q_{t+1}(x_{t+1})}. \tag{3.2}$$

Since the reverse kernels $q_{t|t+1}(x_t|x_{t+1})$ are generally intractable, we can approximate them with parameterized reverse kernels $p^\theta_{t|t+1}(x_t|x_{t+1})$ and use a neural network to learn these approximations. These approximated reverse kernels $p^\theta_{t|t+1}(x_t|x_{t+1})$ are commonly defined through $p^\theta_{0|t+1}$ together with $q_{t|t+1,0}$ [36, 3], resulting in

$$\begin{aligned} p^\theta_{t|t+1} &= \sum_{x_0} q_{t|t+1,0}(x_t|x_{t+1}, x_0) p^\theta_{0|t+1}(x_0|x_{t+1}) \\ &= q_{t+1|t}(x_{t+1}|x_t) \sum_{x_0} \frac{q_{t|0}(x_t|x_0)}{q_{t+1|0}(x_{t+1}|x_0)} p^\theta_{0|t+1}(x_0|x_{t+1}). \end{aligned} \tag{3.3}$$

Generating samples whose distribution $p^\theta(x_0)$ approximately represent the original data distribution $\pi_{\text{data}}(x_0)$ can then be accomplished by sampling the learned generative joint distribution

$$p^\theta_{0:T}(x_{0:T}) = p_{\text{ref}}(x_T) \prod_{t=0}^{T-1} p^\theta_{t|t+1}(x_t|x_{t+1}). \tag{3.4}$$

Figure 3.1 visually summarizes the entire procedure of diffusion models in discrete time, illustrating how x_0 gradually transitions through the noise introduced by each $q_{t+1|t}$ to become completely noisy by time step T. By learning these noise perturbations, we can generate samples by first sampling noise $x_T \sim p_{\text{ref}}(x_T)$ and then iteratively sampling the learned reverse kernels until we approximately yield samples from the true data distribution.

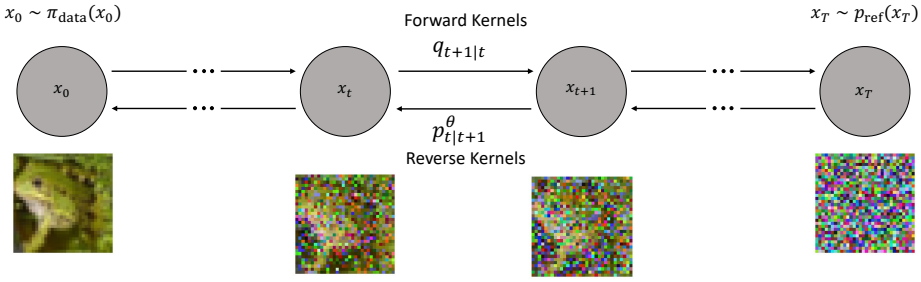


Figure 3.1.: Illustration of the forward and reverse process in a discrete-time diffusion model. The processes are applied to a data sample from the CIFAR-10 dataset [52].

3.2. Design of the Forward Process

In order to attain the desired approximation $q_T(x_T) \approx p_{\text{ref}}(x_T)$ and ensure convergence of the reverse process to the same functional form as the forward process, we need to design the forward process appropriately. For the approximation $q_T(x_T) \approx p_{\text{ref}}(x_T)$, it is essential to select forward kernels $q_{t+1|t}(x_{t+1}|x_t)$ that all admit $p_{\text{ref}}(x_T)$ as a stationary distribution [7]. A stationary distribution is a probability distribution over the states of a Markov process that remains unchanged under the application of the process's transition probabilities [67]. Specifically, if p_{ref} is a stationary distribution of $q_{t+1|t}$, then applying these transition probabilities to p_{ref} will not alter its distribution. This means that for any state $y \in \mathcal{X}$ and any time step $t \in [0, T]$, the probability distribution of the system being in state y at time step $t+1$ is exactly the same, as at time step t. Mathematically, this invariance can be expressed as

$$p_{\text{ref}}(y) = \sum_{z \in \mathcal{X}} p_{\text{ref}}(z) q_{t+1|t}(y|z), \quad \forall y \in \mathcal{X}.$$

For more details on stationary distributions and why choosing p_{ref} as the stationary distribution of $q_{t+1|t}$ can lead to $q_T(x_T) \approx p_{\text{ref}}(x_T)$ for a sufficiently large number of time steps T, refer to [67].

In the case of discrete data, a typical choice for $q_{t+1|t}$ are uniform distributions [38, 3]. The corresponding stationary distribution p_{ref} is the uniform distribution over all states, resulting in a x_T full of noise [7, 22]. The kernels can then be expressed as

$$\text{Uniform:} \qquad q_{t+1|t}(x_{t+1}|x_t) = \delta_{x_{t+1},x_t}(1-\sigma_t) + \frac{(1-\delta_{x_{t+1},x_t})\sigma_t}{S-1}, \tag{3.5}$$

where σ_t is the amount of noise added at timestep t. The amount of noise added at each time step is controlled by a noise schedule. As per definitions in Equations 3.1 and 3.5, we observe that a larger amount of noise added at each timestep accelerates the degradation of the data structures, allowing faster convergence of $q_T(x_T)$ to $p_{\text{ref}}(x_T)$. However, striking a balance between excessive and insufficient noise addition is critical, as it affects not only the forward process but also the reverse process and the model's sampling procedure. On the one hand, by adding too much noise at each time step, the forward and reverse processes do not converge to

the same functional form. This can lead to suboptimal convergence of a model, which cannot adequately recover the original data distribution [93]. On the other hand, adding too little noise leads to a large number of time steps T to ensure convergence to $p_{\text{ref}}(x_T)$. This results in slow sampling speeds as we have to individually step through each time step in the reverse process, as shown in Equation 3.4.

There are several strategies for designing noise schedules to control the amount of noise added at each time step, including linear [36], cosine [65], or exponential [84] functions of the time step. In addition, the noise schedule can also be learned together with the reverse process [47]. These approaches affect perturbations in the forward process differently in terms of speed and magnitude, thereby impacting the overall learning process. For example, in Figure 3.2, we visualize a linear and cosine noise schedule applied alongside uniform forward kernels to a sample from the CIFAR-10 dataset [52].

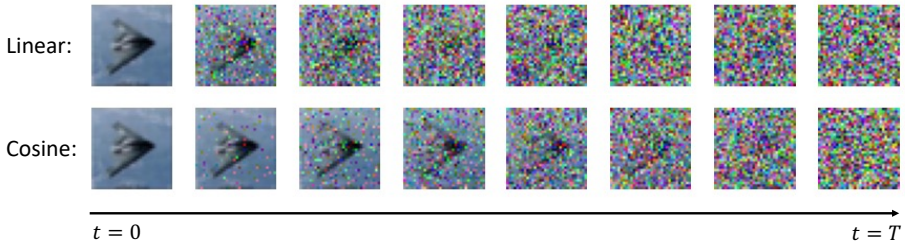


Figure 3.2.: Illustration of linear and cosine noise schedules applied to a data sample from the CIFAR-10 dataset. We selected the number of time steps as $T = 7$ for illustrative purposes, although typical values of T range from 250 to 1000 [36, 3].

While a linear noise schedule applies noise consistently across all time steps, a cosine schedule adjusts the noise level more dynamically. Although both schedules lead to noisy data at time step T, satisfying the condition $q_T(x_T) \approx p_{\text{ref}}(x_T)$, the linear schedule perturbs the data faster than necessary. This results in premature destruction of data structures. In contrast, the cosine schedule preserves more of the data's structure for a longer period of time, potentially leading to more efficient training. Such an improved training process can directly contribute to an enhanced quality of the generated samples, reflecting a more accurate representation of the actual data distribution [65].

3.3. Learning the Reverse Process

In order to generate samples whose distribution $p_0^\theta(x_0)$ is as close as possible to the actual data distribution $\pi_{\text{data}}(x_0)$, we need to learn the parameterization of the reverse kernels $p_{t|t+1}^\theta$. One approach to achieve this is by minimizing the Kullback-Leibler (KL) divergence $\mathcal{D}_{\text{KL}}$ between $p_0^\theta(x_0)$ and $\pi_{\text{data}}(x_0)$ [83]. The KL divergence measures how one probability distribution differs from another, with lower values indicating greater similarity between the distributions [12]. Therefore, by minimizing the KL divergence between $p_0^\theta(x_0)$ and $\pi_{\text{data}}(x_0)$ we can align $p_0^\theta(x_0)$

more closely with the actual data distribution $\pi_{\text{data}}(x_0)$. Mathematically, the KL-divergence between $p_0^\theta(x_0)$ and $\pi_{\text{data}}(x_0)$ can be expressed as follows

$$\begin{aligned}\mathcal{D}_{\text{KL}}\left(\pi_{\text{data}}(x_0) \parallel p_0^\theta(x_0)\right) &= \mathbb{E}_{\pi_{\text{data}}(x_0)}[\log \frac{\pi_{\text{data}}(x_0)}{p_0^\theta(x_0)}] \\ &= \mathbb{E}_{\pi_{\text{data}}(x_0)}[\log \pi_{\text{data}}(x_0)] - \mathbb{E}_{\pi_{\text{data}}(x_0)}[\log p_0^\theta(x_0)].\end{aligned} \tag{3.6}$$

Since the first term in Equation 3.6 does not depend on θ, minimizing the KL divergence is equivalent to maximizing $-\mathbb{E}_{\pi_{\text{data}}(x_0)}[\log p_0^\theta(x_0)]$, which is the expected negative log-likelihood of $p_0^\theta(x_0)$. Yet, directly computing and maximizing the log-likelihood is difficult. It either involves marginalizing out all latent variables $x_{1:T}$,

$$p_0^\theta(x_0) = \sum_{x_{1:T}} p_{0:T}^\theta(x_0, x_{1:T}),$$

or it involves applying the chain rule of probability,

$$p_0^\theta(x_0) = \frac{p_{0:T}^\theta(x_0, x_{1:T})}{p_{1:T|0}^\theta(x_{1:T}|x_0)}.$$

Both approaches are generally intractable for complex models [83]. However, we can derive the discrete-time ELBO L_{DTE} [48, 83], as a tractable upper bound on the negative log-likelihood

$$\mathbb{E}[-\log p_0^\theta(x_0)] \leq \mathbb{E}_{q_{0:T}(x_{0:T})}\left[-\log \frac{p_{0:T}^\theta(x_{0:T})}{q_{1:T|0}(x_{1:T}|x_0)}\right] = L_{\text{DTE}},$$

which can be further simplified to

$$\begin{aligned}L_{\text{DTE}}(\theta) = \mathbb{E}_{\pi_{\text{data}}(x_0)}\Bigg[&\underbrace{\mathcal{D}_{\text{KL}}\left(q_{T|0}(x_T|x_0) \parallel p_{\text{ref}}(x_T)\right)}_{L_T} - \underbrace{\mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p_{0|1}^\theta(x_0|x_1)\right]}_{L_0} \\ &+ \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q_{t+1|0}(x_{t+1}|x_0)}\left[\mathcal{D}_{\text{KL}}\left(q_{t|t+1,0}(x_t|x_{t+1}, x_0) \parallel p_{t|t+1}^\theta(x_t|x_{t+1})\right)\right]}_{L_t}\Bigg].\end{aligned} \tag{3.7}$$

The L_{DTE} loss formulation, as shown in Equation 3.7, consists of three independent terms. This structure simplifies the optimization process, which, in turn, enables effective minimization of the divergence between the model's approximated data distribution $p_0^\theta(x_0)$ and the actual data distribution $\pi_{\text{data}}(x_0)$ [83, 36]. When $q_T(x_T)$ converges to $p_{\text{ref}}(x_T)$, the first term L_T of L_{DTE} will be approximately zero, regardless of the initial data distribution $\pi_{\text{data}}(x_0)$ [3]. For efficient optimization of the other terms, we only need to ensure that the forward kernels $q_{t+1|t}(x_{t+1}|x_t)$ enable the analytical computation of the transition probability $q_{t|0}(x_t|x_0)$ at any arbitrary $t \in [0, T]$ [7]. That allows us to randomly sample time steps t from a uniform distribution $\mathcal{U}(0, T)$, sample x_t from $q_{t|0}(x_t|x_0)$, and optimize individual terms of L_t through stochastic gradient descent [75]. In addition, if the forward kernels admit a tractable computation of $q_{t|t+1,0}(x_t|x_{t+1})$, the KL divergence in L_t can be analytically computed instead of using Monte Carlo estimation [7]. Both properties are satisfied by the commonly used kernels from Equation 3.5.

3.4. Drawbacks of Diffusion Models in Discrete Time

As mentioned in Chapter 1, the discrete-time formulation of diffusion models entails inherent limitations. In a typical discrete-time setting, the training process necessitates selecting a finite number of timesteps T, with noise addition occurring at specific time steps as seen in Equation 3.1. This approach constrains the model's training to learn the denoising exclusively at these preselected timesteps for the learned reverse process, as shown in Equation 3.2. Such a rigid structure in time discretization restricts the model's capability to adapt to different noise levels outside these predetermined points. Consequently, it confines the model to basic ancestral sampling strategies and limits its versatility in handling diverse noise conditions [7, 87].

3.5. Continuous-Time Extension for Continuous Diffusion Models

In the case of continuous state spaces, the mentioned limitations associated with discrete-time diffusion models have been addressed by a formulation of diffusion models within an SDE [49, 68] framework [86]. Instead of noising data at discrete time steps, as seen previously, this approach takes advantage of perturbing data across an infinite range of noise scales. Specifically, the evolution of the perturbed data densities q_t in the forward process is modeled as a solution to an Itô SDE [68], given by

$$\mathrm{d}x = f(x,t)\mathrm{d}t + c(t)\mathrm{d}w,$$

where w represents a Wiener process [68], $f(x,t)$ is a vector-valued drift function, and $c(t)$ is a scalar-valued diffusion coefficient. The corresponding reverse-time SDE can be derived by estimating the score function [41] of the perturbed data densities, resulting in

$$\mathrm{d}x = \Big[f(x,t) - c^2(t)\nabla_x \log q_t(x)\Big]\,\mathrm{d}t + c(t)\mathrm{d}\hat{w}, \tag{3.8}$$

where $\hat{w}$ represents another Wiener process and $\nabla_x \log q_t(x)$ is the score function. The score function is a vector field across the data space pointing towards the modes of the unknown probability density. At each x in the data space, it essentially captures the local change in the probability density, guiding towards regions of higher likelihood [85]. By learning the score function and following it in the reverse process until a mode is reached, we can approximately generate samples from the true data distribution [86].

The continuous-time formulation outlined above enables the learning of denoising at any possible moment within the time interval. This capability enhances the model's adaptability, making it possible to employ more advanced sampling techniques [86, 7]. These samplers have demonstrated their effectiveness in expediting the sampling process [43, 80] and significantly improving the quality of generated samples [86, 18]. However, as defined for continuous data, the score function is not directly applicable in discrete state spaces. The gradient of the log-likelihood with respect to a discrete variable does not exist [40, 87]. This limitation necessitates an alternative approach to leverage the advantages of a continuous-time framework for modeling discrete data directly in discrete state spaces.

4. CTMC Frameworks for Discrete Diffusion Models

The τLDR framework emerged as a response to the challenges encountered when attempting to adapt existing continuous-time frameworks of diffusion models to discrete state spaces [7]. This framework provides an alternative approach based on CTMCs, enabling the representation of state transitions and probabilistic evolution in continuous time without the need for explicit score estimation. Learning within this framework is facilitated by extending the ELBO to the continuous-time domain. Expanding on these principles, the SDDM framework [87] similarly adopts the CTMC formulation of diffusion models. However, instead of using the ELBO for training, SDDM introduces a novel categorical ratio matching loss inspired by a score matching perspective.

In this chapter, we build upon the foundational concepts established in previous chapters to derive these CTMC formulations of diffusion models in discrete state spaces. We start by outlining the shared methodology of both frameworks in formulating diffusion processes as CTMCs. Subsequently, we examine the distinct strategies employed by each framework to learn the continuous-time reverse process. Throughout these derivations, we identify opportunities for novel comparisons and analyses that have not been previously explored. Specifically, we outline potential areas for investigation, including the impact of alternative loss functions on sample quality in both frameworks, and the effects of different parameterizations within the SDDM framework. Detailed comparisons and analyses of these aspects will be conducted in our experimental analysis in Chapter 8.

4.1. Continuous-Time Modeling

The goal is to define a continuous-time process that transitions from an initial distribution $q_0(\boldsymbol{x}_0) = \pi_{\text{data}}(\boldsymbol{x}_0)$ at time $t = 0$ to an easy to sample reference distribution $p_{\text{ref}}(\boldsymbol{x}_T) \approx q_T(\boldsymbol{x}_T)$ at time $t = T$. Consequently, to generate samples, we also need to establish a continuous-time reverse process that brings us back from $p_{\text{ref}}(\boldsymbol{x}_T)$ to $\pi_{\text{data}}(\boldsymbol{x}_0)$. Note that we now consider generating D-dimensional discrete data $\boldsymbol{x}_0 = (x_0^1, \dots, x_0^D) \in \mathcal{X}^D$, where each dimension can take one of $S = |\mathcal{X}|$ possible values.

As proposed in [7], the modeling process is initiated by representing the forward process of our diffusion model as a D-dimensional CTMC $\{\boldsymbol{X}_t\}_{t\in[0,T]}$, with $\boldsymbol{X}_t$ taking values in $\mathcal{X}^D$. At time $t = 0$, we initialize the CTMC with the initial distribution $q_0(\boldsymbol{x}_0) = \pi_{\text{data}}(\boldsymbol{x}_0)$. The infinitesimal transition probabilities of the CTMC are governed by the rate matrices $R_t \in \mathbb{R}^{S^D \times S^D}$, as seen

in Chapter 2:

$$q_{t|t-h}(\boldsymbol{y}|\boldsymbol{x}) = \delta_{\boldsymbol{y},\boldsymbol{x}} + R_t(\boldsymbol{x},\boldsymbol{y})h + o(h).$$

The probabilistic evolution of the forward process can then be described through the corresponding Kolmogorov forward equation [87], denoted as

$$\frac{\partial}{\partial t} q_{t|r}(\boldsymbol{x}_t|\boldsymbol{x}_r) = \sum_{\boldsymbol{y}\in\mathcal{X}^D} q_{t|r}(\boldsymbol{y}|\boldsymbol{x}_r) R_t(\boldsymbol{y},\boldsymbol{x}_t), \qquad r < t. \tag{4.1}$$

Comparing this to the discrete-time case, we can observe that R_t assumes a role analogous to the discrete-time forward kernels $q_{t+1|t}$ in how the forward process is defined [7]. This analogy is key, as choosing an appropriate R_t allows us to guide the marginal distribution $q_T(\boldsymbol{x}_T)$ towards the reference distribution $p_{\text{ref}}(\boldsymbol{x}_T)$ and similarly enables efficient training through analytical computation of $q_{t|0}$ [7]. The details of designing R_t to fulfill these properties are outlined in Chapter 6. This chapter explores the considerations and strategies for efficiently simulating the forward and reverse process, highlighting the important role of an appropriate design of R_t in enabling these simulations.

Considering the reverse process, which transitions from the reference distribution $p_{\text{ref}}(\boldsymbol{x}_T) \approx q_T(\boldsymbol{x}_T)$ back to the initial distribution $\pi_{\text{data}}(\boldsymbol{x}_0)$, we can model it as the time-reversed CTMC $\{\hat{\boldsymbol{X}}_t\}_{t\in[0,T]} = \{\boldsymbol{X}_{T-t}\}_{t\in[0,T]}$, with $q_T(\boldsymbol{x}_T)$ as initial distribution and $\pi_{\text{data}}(\boldsymbol{x}_0)$ as terminal distribution. We obtain a similar equation for the infinitesimal transition probabilities in the reverse process [7], expressed as

$$q_{t|t+h}(\boldsymbol{x}|\boldsymbol{y}) = \delta_{\boldsymbol{x},\boldsymbol{y}} + \hat{R}_t(\boldsymbol{y},\boldsymbol{x})h + o(h),$$

where the reverse rates $\hat{R}_t \in \mathbb{R}^{S^D \times S^D}$ are given by

$$\hat{R}_t(\boldsymbol{x},\boldsymbol{y}) = R_t(\boldsymbol{y},\boldsymbol{x}) \frac{q_t(\boldsymbol{y})}{q_t(\boldsymbol{x})}. \tag{4.2}$$

Consequently, the reverse process can be described by the following Kolmogorov equation [96]

$$\frac{\partial}{\partial r} q_{r|t}(\boldsymbol{x}_r|\boldsymbol{x}_t) = \sum_{\boldsymbol{y}\in\mathcal{X}^D} q_{r|t}(\boldsymbol{y}|\boldsymbol{x}_t) \hat{R}_t(\boldsymbol{y},\boldsymbol{x}_r), \quad r < t.$$

As visually illustrated in Figure 4.1, this CTMC formulation of diffusion models enables the simulation of their processes continuously over time. The procedure begins by sampling $\boldsymbol{x}_0$ from $\pi_{\text{data}}(\boldsymbol{x}_0)$ and continuously introducing noise over the continuous-time interval $[0,T]$ through the forward process characterized by rates R_t. Subsequently, samples can be generated by sampling $\boldsymbol{x}_T$ from $p_{\text{ref}}(\boldsymbol{x}_T)$ and reversing the introduced noise through the continuous reverse process defined by $\hat{R}_t$. Unfortunately, $\hat{R}_t$ is intractable as the marginals $q_t(\cdot) = \sum_{\boldsymbol{x}_0\in\mathcal{X}^D} q_{t|0}(\cdot|\boldsymbol{x}_0)\pi_{\text{data}}(\boldsymbol{x}_0)$ in Equation 4.2 require access to $\pi_{\text{data}}(\boldsymbol{x}_0)$. Therefore, we need to approximate $\hat{R}_t$ with a parameterized reverse rate matrix $\hat{R}_t^\theta$ and learn this approximation to simulate the reverse process for generating samples [7].

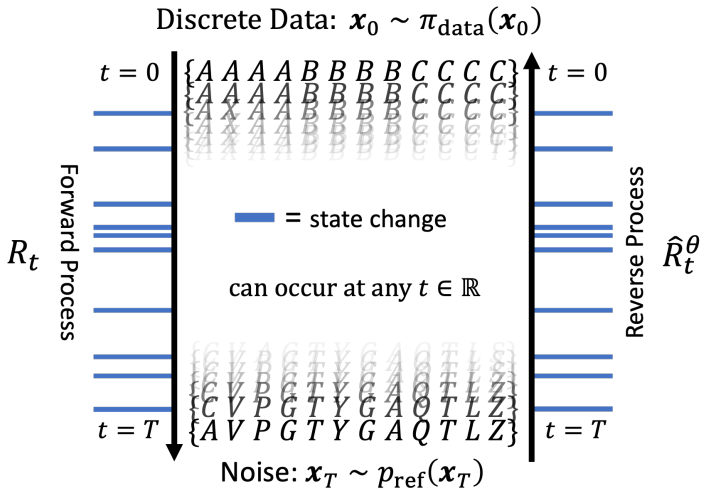


Figure 4.1.: Illustration of the forward and reverse process in a continuous-time diffusion model, adapted from [7]. The processes are applied to a discrete data sample composed of an ordered sequence of letters, serving as an illustrative example.

4.2. Continuous-Time ELBO in the τLDR Framework

In the τLDR framework, the methodology for estimating the reverse rate matrix $\hat{R}_t$ uses a continuous-time extension of the ELBO. The following subsections detail this methodology, starting with the parameterization of $\hat{R}_t$ in τLDR. We then derive the continuous-time formulation of the ELBO to enable learning this parameterization, followed by a discussion of efficient training strategies for this objective. In addition, we examine an existing loss extension of the continuous-time ELBO (CT-ELBO) and propose an alternative loss function for learning the reverse rate matrix, aiming to improve the estimation quality.

4.2.1. Parameterization of the Reverse Rates in τLDR

For the parameterization of $\hat{R}_t$ we can rewrite the marginal ratio as

$$\hat{R}_t(\boldsymbol{x}, \boldsymbol{y}) = R_t(\boldsymbol{y}, \boldsymbol{x}) \frac{q_t(\boldsymbol{y})}{q_t(\boldsymbol{x})} = R_t(\boldsymbol{y}, \boldsymbol{x}) \sum_{\boldsymbol{x}_0} \frac{q_{t|0}(\boldsymbol{y}|\boldsymbol{x}_0)}{q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)} q_{0|t}(\boldsymbol{x}_0|\boldsymbol{x})$$

and consider an approximation of $\hat{R}_t$ with a parameterized denoising model $p^\theta_{0|t}(\boldsymbol{x}_0|\boldsymbol{x})$ [7]. The denoising model outputs categorical probabilities over the state values of $\boldsymbol{x}_0$ conditioned on a noisy $\boldsymbol{x}$. In general, one could choose any parameterization since it is just a question of what the neural network predicts. The decision to use $p^\theta_{0|t}(\boldsymbol{x}_0|\boldsymbol{x})$ in τLDR draws inspiration from the discrete-time and discrete state space framework (D3PM) [3], which demonstrated superior quality of the generated samples with such an approximation of the reverse kernels.

Parameterizing this denoising model then results in an analog equation of the discrete-time Equation 3.3:

$$\hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{y}) = R_t(\boldsymbol{y}, \boldsymbol{x}) \sum_{\boldsymbol{x}_0} \frac{q_{t|0}(\boldsymbol{y}|\boldsymbol{x}_0)}{q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)} p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x}). \tag{4.3}$$

4.2.2. Derivation and Intuition of the Continuous-Time ELBO

The continuous-time formulation of the ELBO can be derived by considering a partition of the continuous time interval $[0, T]$ into an infinite series of noising or denoising steps, where each time interval h between consecutive steps is infinitesimally small [7]. Then, the discrete-time ELBO L_{DTE} naturally transitions into the CT-ELBO. Formally, for the time-reversed CTMC with reverse rate $\hat{R}_t^\theta$, initial distribution $p_{\text{ref}}(\boldsymbol{x}_T)$, and terminal distribution $\pi_{\text{data}}(\boldsymbol{x}_0)$, the CT-ELBO is given by (full derivation in Appendix A.3 [7])

$$L_{\text{CTE}}(\theta) = T\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(\boldsymbol{x})g_t(\boldsymbol{y}|\boldsymbol{x})} \left[\sum_{\boldsymbol{x}'\neq\boldsymbol{x}} \hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{x}') - \mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}) \right] + C, \tag{4.4}$$

where C is a constant independent of θ and

$$\mathcal{Z}_t(\boldsymbol{x}) = \sum_{\boldsymbol{x}'\neq\boldsymbol{x}} R_t(\boldsymbol{x}, \boldsymbol{x}'), \qquad g_t(\boldsymbol{y}|\boldsymbol{x}) = \frac{(1 - \delta_{\boldsymbol{y},\boldsymbol{x}}) R_t(\boldsymbol{x}, \boldsymbol{y})}{\mathcal{Z}_t(\boldsymbol{x})}.$$

Here, $g_t(\boldsymbol{y}|\boldsymbol{x})$ denotes the probability of transitioning from $\boldsymbol{x}$ to $\boldsymbol{y}$ given that a transition occurs at time t. We additionally sample an auxiliary $\boldsymbol{y}$ from this distribution g_t instead of computing the exact expectation over $\boldsymbol{y}$, as it reduces the number of neural network evaluations for computing the CT-ELBO [7]. The sampled $\boldsymbol{y}$ from this distribution differs from $\boldsymbol{x}$ in only a single dimension. More details on this process can be found in the derivation of the CT-ELBO in Appendix A.3.

The intuition behind L_{CTE} can be further understood by delving into its similarities with L_{DTE}. When we consider very small time intervals h between two consecutive noising or denoising steps in the discrete-time formulation, only the KL term of L_{DTE} (refer to Equation 3.7), denoted as

$$-\mathbb{E}_{q_t(\boldsymbol{x}_t)q_{t+1|t}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)} \left[\log p_{t|t+1}^\theta(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}) \right], \tag{4.5}$$

will be nonzero when L_{DTE} transitions to L_{CTE} in the limit $h \to 0$. Minimizing the term in Equation 4.5 implies sampling pairs $(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ from the forward process and then maximizing the probability assigned by the model for their reverse transition [7]. For the interpretation of L_{CTE}, we can use a similar approach by first expressing $\log p_{t|t+1}^\theta(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})$ from Equation 4.5 in terms of $\hat{R}_t^\theta$ as follows

$$\begin{aligned} \log p_{t|t+1}^\theta(\boldsymbol{x}_t|\boldsymbol{x}_{t+1}) = \delta_{\boldsymbol{x}_t,\boldsymbol{x}_{t+1}} \left[\hat{R}_t^\theta(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t) h + o(h) \right] \\ + (1 - \delta_{\boldsymbol{x}_t,\boldsymbol{x}_{t+1}}) \log \left[\hat{R}_t^\theta(\boldsymbol{x}_{t+1}, \boldsymbol{x}_t) h + o(h) \right]. \end{aligned} \tag{4.6}$$

Here, we distinguish the cases when $\boldsymbol{x}_t = \boldsymbol{x}_{t+1}$ and when $\boldsymbol{x}_t \neq \boldsymbol{x}_{t+1}$ (for details see full proof of CT-ELBO). In the limit $h \to 0$, the first term of Equation 4.6 becomes the term

$\sum_{\boldsymbol{x}' \neq \boldsymbol{x}} \hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{x}')$ in L_{CTE}, while the second term becomes $\mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})$ [7]. Now, in the pursuit of minimizing L_{CTE}, we sample $(\boldsymbol{x}, \boldsymbol{y})$ from the forward process and then maximize the model's assigned probability for the reverse direction, similar to L_{DTE}. The added complexity arises from our consideration of the cases where $\boldsymbol{x}_t = \boldsymbol{x}_{t+1}$ and $\boldsymbol{x}_t \neq \boldsymbol{x}_{t+1}$. When $\boldsymbol{x}_t = \boldsymbol{x}_{t+1}$, corresponding to $\sum_{\boldsymbol{x}' \neq \boldsymbol{x}} \hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{x}')$, this term focuses on minimizing the reverse rate out of state $\boldsymbol{x}$, interpreted as maximizing the probability of no transition occurring. Conversely, when $\boldsymbol{x}_t \neq \boldsymbol{x}_{t+1}$, corresponding to the term $\mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})$ in L_{CTE}, the focus is on maximizing the reverse rate from $\boldsymbol{y}$ to $\boldsymbol{x}$, hence maximizing the probability of a transition occurring [7].

4.2.3. Efficient Training with the Continuous-Time ELBO

Although the construction of the distribution $g_t(\boldsymbol{y}|\boldsymbol{x})$ represents a significant step in reducing the computational load for evaluating L_{CTE}, there is still the need for two separate forward passes through the neural network [7]. One pass is required to calculate $\hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{x}')$ using $p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x})$, and another is needed to determine $\hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})$ using $p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{y})$. However, this is not very efficient, considering that $\boldsymbol{y}$ is generated from $\boldsymbol{x}$ by altering only a single dimension in the forward transition [7]. As a result, $\boldsymbol{y}$ and $\boldsymbol{x}$ are closely related, and we can feasibly approximate samples drawn from $\boldsymbol{x} \sim q_t(\boldsymbol{x})$ by using samples from $\boldsymbol{y} \sim \sum_{\boldsymbol{x}} q_t(\boldsymbol{x}) g_t(\boldsymbol{y}|\boldsymbol{x})$. This approximation is valid, since $q_t(\boldsymbol{x})$ and $\boldsymbol{y} \sim \sum_{\boldsymbol{x}} q_t(\boldsymbol{x}) g_t(\boldsymbol{y}|\boldsymbol{x})$ are very similiar distributions (see proof in Appendix A.4 [7]). Exploiting this similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$ streamlines the process to only one forward pass and leads to a more efficient form of this loss, given by

$$L_{\mathrm{eCTE}}(\theta) = T \mathbb{E}_{t \sim \mathcal{U}(0,T) q_t(\boldsymbol{x}) g_t(\boldsymbol{y}|\boldsymbol{x})} \left[\sum_{\boldsymbol{x}' \neq \boldsymbol{y}} \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}') - \mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}) \right]. \tag{4.7}$$

Furthermore, ablation studies in [7] have demonstrated that training models within τLDR using the simplified loss L_{eCTE} not only reduces the computational complexity, but also leads to an improvement in the quality of generated samples compared to using L_{CTE} for training.

To efficiently optimize the simplified loss L_{eCTE}, we can employ arbitrary neural network architectures and update the parameters θ using stochastic gradient descent. As illustrated in Pseudo-algorithm 1, the full training process involves sampling a batch of data points $\boldsymbol{x}_0$ from $\pi_{\mathrm{data}}(\boldsymbol{x}_0)$, noise each data point until a randomly chosen time $t \sim \mathcal{U}(0, T)$ through analytical computation of $q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)$. Subsequently, we sample an auxiliary $\boldsymbol{y}$ from $g_t(\boldsymbol{y}|\boldsymbol{x})$ for each $\boldsymbol{x}$ and compute the reverse rates with a single forward pass, as demonstrated in Equation 4.7. Finally, we evaluate L_{eCTE} and update the parameters using stochastic gradient descent. This procedure is repeated until we reach a predefined stopping criterion.

4.2.4. Loss Extensions in τLDR

Apart from the ELBO, another commonly employed auxiliary loss function in discrete time is the negative log-likelihood of $p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x})$, which measures the reconstruction quality of the original data $\boldsymbol{x}_0$ from its corrupted version $\boldsymbol{x}$ [97]. The mathematical expression for this loss

Algorithm 1 Training with L_{eCTE}

repeat
  $\boldsymbol{x}_0 \sim \pi_{\text{data}}(\boldsymbol{x}_0)$
  $t \sim \mathcal{U}(0, T)$
  $\boldsymbol{x} \sim q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)$
  $\boldsymbol{y} \sim g_t(\boldsymbol{y}|\boldsymbol{x})$
  Take gradient step on:
  $\nabla_\theta \left[\sum_{\boldsymbol{x}' \neq \boldsymbol{y}} \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}') - \mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})\right]$
until converged

function is given by

$$L_{\text{ll}}(\theta) = T\mathbb{E}_{t\sim\mathcal{U}(0,T)\pi_{\text{data}}(\boldsymbol{x}_0)q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)}\left[-\log p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x})\right]. \tag{4.8}$$

Note that both L_{ll} in Equation 4.8 and L_{DTE} in Equation 3.7 share the same global minima with $p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x})$ being the true posterior $q_{0|t}(\boldsymbol{x}_0|\boldsymbol{x})$ [97]. However, both losses have different optimization landscapes, and it is unknown which loss is easier to minimize [60, 15]. Nevertheless, in the discrete-time formulation in [3], an improvement in the quality of generated samples was observed when combining L_{DTE} with L_{ll} resulting in the loss function $L_{\text{DTEll}} = L_{\text{DTE}} + \lambda L_{\text{ll}}$, where λ is a hyperparameter typically set between 0.001 and 0.01 [3].

Following this insight, the τLDR framework adopts a similar approach by combining L_{eCTE} with L_{ll}, resulting in

$$\begin{aligned} L_{\text{CTEll}}(\theta) &= T\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(\boldsymbol{x})g_t(\boldsymbol{y}|\boldsymbol{x})}\left[\sum_{\boldsymbol{x}'\neq\boldsymbol{y}} \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}') - \mathcal{Z}_t(\boldsymbol{x}) \log \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})\right] \\ &\quad + \lambda T\mathbb{E}_{t\sim\mathcal{U}(0,T)\pi_{\text{data}}(\boldsymbol{x}_0)q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)g_t(\boldsymbol{y}|\boldsymbol{x})}\left[-\log p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{y})\right] + C \\ &= L_{\text{eCTE}} + \lambda L_{\text{ll}}, \end{aligned}$$

Training then follows the same procedure as in Pseudo-algorithm 1, but with taking the gradient step on L_{CTEll} instead of L_{eCTE}.

However, in contrast to the discrete-time case, an empirical investigation of training models solely with L_{ll} in the continuous-time setting is not provided in [7]. Therefore, we extend our analysis of the τLDR framework by training models only with this objective and comparing their performance in terms of sample quality to models trained with L_{CTEll}.

4.3. Categorical Ratio Matching in the SDDM Framework

Instead of employing the CT-ELBO to learn the reverse rate matrix, the SDDM framework uses a score-based learning approach. In this section, we detail this approach, starting with the rationale for adopting a score-based loss. Following this, we extend score matching to general categorical data, elaborate on the necessary parameterizations of the reverse rates

within SDDM, and introduce the categorical ratio matching loss for learning these parameterizations. Additionally, we briefly discuss architectural constraints when using the categorical ratio matching loss for training. Moreover, we adapt the CT-ELBO for use within the SDDM framework and propose an alternative loss that combines the CT-ELBO with the categorical ratio matching loss to improve sample quality. In addition, we introduce another alternative loss function that leverages a similar auxiliary denoising objective, as shown in Equation 4.8.

4.3.1. Motivation behind a Score-Based Learning Approach

Although training diffusion models with the ELBO as an approximation to maximum likelihood estimation has achieved great success [17, 36], score matching demonstrated superior sample quality in continuous state spaces [86]. The general problem with maximizing likelihood is when we want to learn an arbitrary unknown probability distribution $\pi(\boldsymbol{x})$, which can be written in the parameterized form

$$p^\theta(\boldsymbol{x}) = \frac{e^{-\phi^\theta(\boldsymbol{x})}}{Z^\theta}, \tag{4.9}$$

where $\phi^\theta(\boldsymbol{x})$ is an arbitrarily flexible function, we need to compute the normalization constant $Z^\theta = \int e^{-\phi^\theta(\boldsymbol{x})} \mathrm{d}\boldsymbol{x}$, to ensure probabilities between 0 and 1 [83]. However, calculating the normalization constant in practice is intractable [34]. In order to avoid computing Z^θ, likelihood-based methods often have to use a surrogate loss like the ELBO, as we saw previously.

Score matching instead avoids the need to compute Z^θ and bypasses loss approximations [41]. The core idea of score matching is that instead of learning the unknown probability distribution $\pi(\boldsymbol{x})$, we learn the score function

$$\nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x}).$$

Note that although the score function does not represent the data distribution in its original form, it still contains important information about this distribution. In particular, as mentioned in Chapter 3, the score function points towards regions of higher probability density within the data space. In the context of diffusion models, this property is used to generate samples by following the direction indicated by the score functions during the reverse process, as seen in Equation 3.8.

Estimating the score function is motivated by the fact that by taking the gradient with respect to $\boldsymbol{x}$ of the log of Equation 4.9, we avoid computing the normalization constant, leading to the following expression

$$\begin{aligned} \nabla_{\boldsymbol{x}} \log p^\theta(\boldsymbol{x}) &= -\nabla_{\boldsymbol{x}} \phi^\theta(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log Z^\theta \\ &= -\nabla_{\boldsymbol{x}} \phi^\theta(\boldsymbol{x}). \end{aligned}$$

We can approximate the score function with a neural network and learn the network parameters by matching the learned score function $\nabla_{\boldsymbol{x}} \log p^\theta(\boldsymbol{x})$ with the actual score function $\nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x})$ through minimizing the Fisher divergence between them [41, 84]:

$$\mathbb{E}_{\pi(\boldsymbol{x})} \left[\left\| \nabla_{\boldsymbol{x}} \log p^\theta(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x}) \right\|_2^2 \right]. \tag{4.10}$$

4.3.2. From Score Matching to Categorical Ratio Matching

Motivated by the advantages of score matching, the methodology in SDDM extends a score matching approach to continuous time and discrete state spaces [87]. Nevertheless, as previously mentioned, the problem with deriving an analog of a score matching loss for discrete data is that the score function is not defined for discrete random variables. To address this limitation, the work in [40] proposed ratio matching as an extension of score matching to binary data. The basic idea is that instead of focusing on gradients, ratio matching focuses on the ratios of probabilities between adjacent discrete states $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$. In particular, these adjacent states have only a flip in the d-th dimension [40], expressed as

$$\tilde{\boldsymbol{x}} = (x^1, x^2, \ldots, 1 - x^d, \ldots, x^D).$$

The shift in focus is inspired by the underlying principle of score matching. It aims to capture local changes in probability density functions by aligning the gradient of the model's log-likelihood $\nabla_{\boldsymbol{x}} \log p^\theta(\boldsymbol{x})$ with that of the data's log-likelihood $\nabla_{\boldsymbol{x}} \log \pi(\boldsymbol{x})$, as seen in Equation 4.10. Ratio matching adapts to this by instead comparing the ratios of probabilities between adjacent states, effectively capturing a comparable "local change" in the probability mass function [40]:

$$\mathbb{E}_{\pi(\boldsymbol{x})}\left[\left\|\frac{p^\theta(X^d = x^d, \boldsymbol{x}^{\setminus d})}{p^\theta(X^d = 1 - x^d, \boldsymbol{x}^{\setminus d})} - \frac{\pi(X^d = x^d, \boldsymbol{x}^{\setminus d})}{\pi(X^d = 1 - x^d, \boldsymbol{x}^{\setminus d})}\right\|_2^2\right]. \tag{4.11}$$

In doing so, ratio matching preserves the spirit of score matching and provides a practical method for accurately estimating probability distributions in discrete state spaces [40]. Note that in Equation 4.11, we overload the notation to use $(X^d = 1 - x^d, \boldsymbol{x}^{\setminus d})$, where $\boldsymbol{x}^{\setminus d}$ represents all dimensions of $\boldsymbol{x}$ excepth the d-th dimension, to denote $(x^1, x^2, \ldots, X^d = 1 - x^d, \ldots, x^D)$ such that only the value of the d-th random variable X^d is replaced.

Motivated by learning the binary ratio of probabilities as an extension to score matching, as shown in Equation 4.11, the methodology in SDDM aims to learn the ratio $\frac{q_t(\boldsymbol{x})}{q_t(\boldsymbol{y})}$ in the reverse rate equation

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

in a similar manner. To further contextualize this approach, the ratio $\frac{q_t(\boldsymbol{x})}{q_t(\boldsymbol{y})}$ has basically the same behaviour as the score function in Equation 3.8 [87]. Once the ratio is known, one can obtain the generative flow towards the initial distribution $\pi_{\text{data}}(\boldsymbol{x}_0)$ [87]. However, binary ratio matching must be generalized to the categorical case to learn this ratio. The generalization relies on matching the singleton conditional distribution in categorical spaces

$$\pi(X^d = s|\boldsymbol{x}^{\setminus d}) = \frac{\pi(X^d = s, \boldsymbol{x}^{\setminus d})}{\sum_{s' \in \mathcal{X}} \pi(X^d = s', \boldsymbol{x}^{\setminus d})}, \tag{4.12}$$

where the singleton conditional distribution $\pi(X^d = s|\boldsymbol{x}^{\setminus d})$ represents the probability of observing a particular state value $s \in \mathcal{X}$ in the d-th dimension, given all other dimensions of $\boldsymbol{x}$ excluding the d-th dimension [62]. This approach is based on the principle that matching this conditional distribution from Equation 4.12 is sufficient due to the property that a joint

probability distribution π of random variables $X^1, X^2, \ldots, X^D$ is completely determined by its singleton conditional distributions [6]. Consequently, a probability ratio of joint distributions can be decomposed into a product entirely determined by their singleton conditional distributions [87]:

$$\begin{aligned}\frac{\pi(\boldsymbol{y})}{\pi(\boldsymbol{x})} &= \frac{\pi(y^1, y^2, y^3, \ldots, y^{D-1}, y^D)}{\pi(x^1, x^2, x^3, \ldots, x^{D-1}, x^D)} \\ &= \frac{\pi(y^1, y^2, y^3, \ldots, y^{D-1} y^D)}{\pi(x^1, y^2, y^3, \ldots, y^{D-1}, y^D)} \frac{\pi(x^1, y^2, y^3, \ldots, y^{D-1} y^D)}{\pi(x^1, x^2, y^3, \ldots, y^{D-1}, y^D)} \cdots \frac{\pi(x^1, x^2, x^3, \ldots, x^{D-1} y^D)}{\pi(x^1, x^2, x^3, \ldots, x^{D-1}, x^D)} \\ &= \prod_{d=1}^{D} \frac{\pi(X^d = y^d | \boldsymbol{x}^{1:d-1}, \boldsymbol{y}^{d+1:D})}{\pi(X^d = x^d | \boldsymbol{x}^{1:d-1}, \boldsymbol{y}^{d+1:D})}.\end{aligned}$$

Hence, if two distributions π_1 and π_2 have identical singleton conditional distributions, then the relation

$$\frac{\pi_1(\boldsymbol{y})}{\pi_1(\boldsymbol{x})} = \frac{\pi_2(\boldsymbol{y})}{\pi_2(\boldsymbol{x})}.$$

holds for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}^D$ [87]. This result implies for us that we can learn the probability ratio $\frac{q_t(\boldsymbol{y})}{q_t(\boldsymbol{x})}$ by employing a neural network $p_t^\theta(\cdot|\cdot)$ to match the conditional distributions [87]:

$$p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d}) \approx q_t(X_t^d | \boldsymbol{x}^{\setminus d}) \to \frac{q_t(X_t^d = y^d, \boldsymbol{x}^{\setminus d})}{q_t(X_t^d = x^d, \boldsymbol{x}^{\setminus d})} = \frac{q_t(X_t^d = y^d | \boldsymbol{x}^{\setminus d})}{q_t(X_t^d = x^d | \boldsymbol{x}^{\setminus d})} \approx \frac{p_t^\theta(X_t^d = y^d | \boldsymbol{x}^{\setminus d})}{p_t^\theta(X_t^d = x^d | \boldsymbol{x}^{\setminus d})},$$

where X_t^d denotes the d-th dimension at time t of the random variable $\boldsymbol{X}$. From this point on in our work, we will use the shorthand notation $p_t^\theta(y^d | \boldsymbol{x}^{\setminus d})$ instead of $p_t^\theta(X_t^d = y^d | \boldsymbol{x}^{\setminus d})$ if it does not create ambiguity. Further, we will follow the notation in [87] and use $p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d})$ to describe the conditional probability distribution over all possible values $s \in \mathcal{X}$ the random variable X_t^d can take.

4.3.3. Parameterization of the Reverse Rates in SDDM

The conditional parameterization $p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d})$ leads to the following parameterized reverse rates

$$\hat{R}_t^{\theta,d}(\boldsymbol{x}, y^d) = R_t^d(y^d, x^d) \frac{p_t^\theta(y^d | \boldsymbol{x}^{\setminus d})}{p_t^\theta(x^d | \boldsymbol{x}^{\setminus d})}, \tag{4.13}$$

where $\hat{R}_t^{\theta,d}$ is the d-th dimension of the reverse rate matrix $\hat{R}_t^\theta$. Note that this conditional parameterization in SDDM implicitly assumes a dimensional factorization of the rate matrices and, thus, a factorization of the forward and reverse processes [87]. We will further detail this factorization in Chapter 6.

Alternatively, instead of using a neural network to directly model $p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d})$ to estimate the reverse rates, we can adopt an implicit parameterization of $p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d})$, similar to the approach used in the τLDR framework [87]. In τLDR, we implicitly parameterize the ratio $\frac{q_t(\boldsymbol{y})}{q_t(\boldsymbol{x})}$ by using

$p^\theta_{0|t}(\boldsymbol{x}_0|\boldsymbol{x})$ and multiplying it with $q_{t|0}$, as shown in Equation 4.3. In SDDM, we can similarly parameterize $q_t(X^d_t|\boldsymbol{x}^{\setminus d})$ implicitly by writing

$$q_t(X^d_t|\boldsymbol{x}^{\setminus d}) = \sum_{x^d_0} q_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d}) q_{t|0}(X^d_t|x^d_0),$$

and since $q_{t|0}$ is tractable, we can replace the explicit approximation of $q_t(X^d_t|\boldsymbol{x}^{\setminus d})$ by the following implicit approximation

$$p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d}) = \sum_{x^d_0} p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d}) q_{t|0}(X^d_t|x^d_0). \tag{4.14}$$

Consequently, instead of explicitly modeling $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$, we can model it implicitly by using the neural network to approximate $p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d})$, which represents the probability of observing the actual state of $\boldsymbol{x}_0$ in dimension d, given $\boldsymbol{x}^{\setminus d}$. This approach offers us a tractable way to calculate $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$ from $p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d})$ and allows the continued use of the approximation in Equation 4.13. Furthermore, as we will see in Chapter 6, the parameterization of $p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d})$ enables the use of an additional method to generate samples.

In the absence of a comparative evaluation in [87] between the implicit modeling approach of computing $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$ by $p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d})$ with the explicit approach of directly predicting $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$, we conduct an empirical evaluation in our experiments to determine which parameterization leads to a superior quality of the generated samples within the SDDM framework.

For the sake of simplicity in the following discussions, we adopt $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$ as our standard notation. This approach allows us to avoid detailing the implicit parameterization, $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d}) = \sum_{x^d_0} p^\theta_{0|t}(x^d_0|\boldsymbol{x}^{\setminus d}) q_{t|0}(X^d_t|x^d_0)$, for each occurrence. By choosing a consistent notation, we aim to maintain a clear and straightforward discussion, focusing on the conceptual rather than the notational complexities.

4.3.4. Derivation of the Categorical Ratio Matching Loss

To estimate the reverse rates in SDDM, we learn the parameterization of the conditional probability distributions $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$ by minimizing the cross entropy [12, 63] between $p^\theta_t(X^d_t|\boldsymbol{x}^{\setminus d})$ and $q_t(X^d_t|\boldsymbol{x}^{\setminus d})$ [87]. Cross entropy is a central concept in information theory that measures the similarity between probability distributions. Mathematically, the cross entropy $H_{\mathrm{CE}}(q(\boldsymbol{x}), p(\boldsymbol{x}))$ between two probability distributions $q(\boldsymbol{x})$ (the true distribution) and $p(\boldsymbol{x})$ (the predicted distribution) is defined as

$$H_{\mathrm{CE}}(q(\boldsymbol{x}), p(\boldsymbol{x})) = \mathbb{E}_{q(\boldsymbol{x})}\left[-\log p(\boldsymbol{x})\right] = -\sum_{\boldsymbol{x}\in\mathcal{X}^D} q(\boldsymbol{x}) \log p(\boldsymbol{x}),$$

where minimizing $H_{\mathrm{CE}}(q(\boldsymbol{x}), p(\boldsymbol{x}))$ directly corresponds to optimizing the model's predictions to mirror the actual data distribution [27].

Applying this concept to match the conditional distributions $p^\theta_t(s|\boldsymbol{x}^{\setminus d})$ and $q_t(s|\boldsymbol{x}^{\setminus d})$ for all $s \in \mathcal{X}$, for all $d = 1, \ldots, D$, and for all $t \in [0, T]$, translates to matching the expected cross

entropy between these conditionals along the forward process [87]:

$$L_{\text{CRMi}}(\theta) = \int_0^T \sum_{\boldsymbol{x} \in \mathcal{X}^D} q_t(\boldsymbol{x}) \left[\sum_{d=1}^{D} \left(- \sum_{s \in \mathcal{X}} q_t(s|\boldsymbol{x}^{\setminus d}) \log p_t^\theta(s|\boldsymbol{x}^{\setminus d}) \right) \right] \mathrm{d}t, \tag{4.15}$$

where the loss for each $t \in [0, T]$ is minimized if $p_t^\theta(s|\boldsymbol{x}^{\setminus d}) = q_t(s|\boldsymbol{x}^{\setminus d})$ for all $s \in \mathcal{X}$ and for all $d = 1, \ldots, D$. This loss function effectively matches the ratio in Equation 4.12, which is why it is called categorical ratio matching [87]. It encapsulates the process of estimating the probability of observing a particular state value $s \in \mathcal{X}$ in the d-th dimension of $\boldsymbol{x}$ at time t, given all other dimensions of $\boldsymbol{x}$ except the d-th.

Nevertheless, directly computing $q_t(s|\boldsymbol{x}^{\setminus d})$ in Equation 4.15 is intractable, posing challenges in training [87]. Fortunately, due to the specific parameterization, where the conditional distributions do not depend on the dimension d, the term $q_t(s|\boldsymbol{x}^{\setminus d}) \log p_t^\theta(s|\boldsymbol{x}^{\setminus d})$ becomes independent of x^d. This independence allows us to further simplify the term, resulting in a more streamlined loss function (see proof in Appendix A.5 [87])

$$L_{\text{CRM}}(\theta) = \int_0^T \sum_{\boldsymbol{x} \in \mathcal{X}^D} q_t(\boldsymbol{x}) \left[\sum_{d=1}^{D} - \log p_t^\theta(x^d|\boldsymbol{x}^{\setminus d}) \right] \mathrm{d}t. \tag{4.16}$$

The simplicity of the categorical ratio matching loss in Equation 4.16 is derived from the specific structure of the conditional marginal distribution $p_t^\theta(x^d|\boldsymbol{x}^{\setminus d})$. However, this structure also imposes architectural constraints on the neural network [87]. The crucial constraint in the network design is that the predictions for a specific dimension d must be independent of its current state x^d. Any dependence on x^d would result in information leakage, rendering L_{CRM} trivial to solve [87]. To address these architectural considerations, Chapter 5 introduces a variety of network architectures designed to satisfy this constraint. These network architectures enable us to utilize the L_{CRM} in our experimental chapter within the SDDM framework. When using one of these network architectures, the training process associated with L_{CRM}, as outlined in Pseudo-algorithm 2, begins by sampling the training data $\boldsymbol{x}_0$ and adding noise up to a randomly sampled time point t. Next, we predict the probability of the true state value in each dimension d of $\boldsymbol{x}$ by feeding the model the state values of $\boldsymbol{x}$ in all dimensions other than d. Finally, we update the parameters with a gradient step based on the evaluation of L_{CRM}. This iterative process continues until we reach a predefined stopping criterion.

Algorithm 2 Training with L_{CRM}

repeat
- $\boldsymbol{x}_0 \sim \pi_{\text{data}}(\boldsymbol{x}_0)$
- $t \sim \mathcal{U}(0, T)$
- $\boldsymbol{x} \sim q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)$
- Take gradient step on:
- $\nabla_\theta \left[\sum_{d=1}^{D} - \log p_t^\theta(x^d|\boldsymbol{x}^{\setminus d}) \right]$

until converged

4.3.5. Loss Extensions in SDDM

The specific structure of the conditional marginals in SDDM offers the possibility of using L_{CRM} to approximate the reverse rates. However, although it was not mentioned in [87], the conditional parameterization also allows for the continued use of the CT-ELBO loss objective. The adaptability of the CT-ELBO stems from its exclusive reliance on the reverse rates and not on their parameterization. This observation leads to the idea that instead of combining the CT-ELBO with the auxiliary loss L_{ll}, we could combine it with L_{CRM} in SDDM. To implement this, we compute the reverse rates for every dimension with the conditional parameterization

$$\hat{R}_t^{\theta,d}(\boldsymbol{x}, y^d) = R_t^d(y^d, x^d)\frac{p_t^\theta(y^d|\boldsymbol{x}^{\setminus d})}{p_t^\theta(x^d|\boldsymbol{x}^{\setminus d})}.$$

and incorporate these rates into our CT-ELBO calculation

$$L_{\text{eCTE}}(\theta) = T\mathbb{E}_{t\sim\mathcal{U}(0,T)q_t(\boldsymbol{x})g_t(\boldsymbol{y}|\boldsymbol{x})}\left[\sum_{\boldsymbol{x}'\neq\boldsymbol{y}} \hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x}') - \mathcal{Z}_t(\boldsymbol{x})\log\hat{R}_t^\theta(\boldsymbol{y}, \boldsymbol{x})\right].$$

Simultaneously, we utilize the conditional parameterization and employ the categorical ratio matching loss (refer to Equation 4.16). By combining these two objectives, we yield the following loss function

$$L_{\text{CTEcrm}} = L_{\text{eCTE}} + \lambda L_{\text{CRM}}. \tag{4.17}$$

As an additional alternative loss function, we propose to leverage the implicit parameterization of the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d}) = \sum_{x_0^d} p_{0|t}^\theta(x_0^d|\boldsymbol{x}^{\setminus d})q_{t|0}(X_t^d|x_0^d)$ and formulate a similar auxiliary denoising objective

$$L_{\text{ll}^*}(\theta) = T\mathbb{E}_{t\sim\mathcal{U}(0,T)\pi_{\text{data}}(\boldsymbol{x}_0)q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)}\left[-\log p_{0|t}^\theta(\boldsymbol{x}_0|\boldsymbol{x}^{\setminus d})\right].$$

Then, by combining L_{ll^*} with the categorical ratio matching loss, we yield the alternative loss

$$L_{\text{CRMll}} = L_{\text{CRM}} + \lambda L_{\text{ll}^*}. \tag{4.18}$$

In the experimental chapter, we leverage this versatility and additionally train our models in the SDDM framework with the loss functions from Equations 4.17 and 4.18. This approach allows us to assess the impact of three different loss functions, L_{CRM}, L_{CTEcrm}, and L_{CRMll}, on models with consistent architectures and parameterizations within the SDDM framework. Our goal is to determine whether training models with our custom loss functions can improve the quality of generated samples compared to training with L_{CRM}.

4.4. Comparative Summary of τLDR and SDDM

The τLDR and SDDM frameworks share a common foundation in the definition and characterization of the forward and reverse processes. The forward process is represented as a CTMC defined by an initial distribution q_0 and a transition rate matrix R_t. Conversely, the reverse process corresponds to the time-reversed CTMC, characterized by an initial distribution q_T

and a reverse rate matrix $\hat{R}_t$. However, the reverse rates are unknown and must be estimated to generate samples. The fundamental difference between these two frameworks lies in their approaches for training the models to estimate the reverse rates.

In the τLDR framework, learning is done by extending the discrete-time ELBO to the continuous-time domain. This extension is derived by partitioning the continuous-time interval into an infinite series of infinitesimally small time steps, where the discrete-time ELBO seamlessly transitions into its continuous-time formulation. The ELBO itself is an upper bound on the negative log-likelihood of $p_0^\theta(\boldsymbol{x}_0)$, facilitating the training process by optimizing a surrogate objective that approximates the true data distribution $\pi_{\text{data}}(\boldsymbol{x}_0)$.

In contrast, the SDDM loss objective L_{CRM} follows a score matching or ratio matching perspective. It directly fits the rates in the time-reversed CTMC by reformulating the unknown ratio $\frac{q_t(\boldsymbol{y})}{q_t(\boldsymbol{x})}$ in the reverse rate equation as a ratio of conditional marginals $q_t(X_t^d|\boldsymbol{x}^{\setminus d})$. To learn an approximation of the conditional distributions $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$, it uses cross-entropy minimization between $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$ and $q_t(X_t^d|\boldsymbol{x}^{\setminus d})$. Notably, this conditional parameterization in the SDDM framework also allows the continued use of CT-ELBO loss to train the models.

Efficiently training models with the CT-ELBO in τLDR involves several optimization techniques. First, we introduced $g_t(\boldsymbol{y}|\boldsymbol{x})$ to reduce the number of neural network evaluations to two. To further improve efficiency, we apply the "one forward pass trick", which allows a single evaluation of the neural network by approximating $\boldsymbol{x}$ with an auxiliary $\boldsymbol{y}$.

In comparison, to evaluate L_{CRM}, it suffices to sample an $\boldsymbol{x}$ from $q_{t|0}(\boldsymbol{x}|\boldsymbol{x}_0)$, conceal the d-th dimension of $\boldsymbol{x}$, and then predict the logits of that dimension d, given all other dimensions of $\boldsymbol{x}$. This allows us to evaluate this loss function directly with just one forward pass and no need for an approximation.

However, while in τLDR, we have the flexibility to choose any neural network parameterization, in SDDM, the simplicity of the L_{CRM} loss constrains the neural network architecture.

5. Neural Network Architectures for Using the Categorical Ratio Matching Loss

This chapter presents three distinct neural network architectures proposed in [87] to address the architectural challenges associated with using L_{CRM} for training in SDDM. These architectures can model the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$ while maintaining independence from x^d. However, they differ in their tradeoffs between model flexibility and computational cost. The variations in these architectures offer different approaches to balance these critical aspects in model design. In our experimental analysis, we employ two of them to effectively use L_{CRM} and subsequently compare their performance.

For the sake of generality, the following discussion considers an explicit parameterization of $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$. Nonetheless, the same designs can be applied to the implicit parameterization of $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$ [87].

5.1. Energy Based Models

Energy-based models (EBMs) [56, 89, 53] represent a highly general and flexible form of modeling the conditional marginals. Unlike traditional probabilistic models that directly model the data probability distribution, EBMs define a probability distribution indirectly through an energy function $f^\theta(\boldsymbol{x}, t) : \mathcal{X}^D \times \mathbb{R} \to \mathbb{R}$, which is parametrized by a neural network. In EBMs, each state is associated with an energy value, where lower energy values correspond to more likely states, and higher energy values indicate less likely ones. This approach embeds a probabilistic understanding directly into the model's architecture. By using an EBM, we can model the conditional marginals with any neural network architecture as follows

$$p_t^\theta(s|\boldsymbol{x}^{\setminus d}) = \frac{\exp\left(-f^\theta([s, \boldsymbol{x}^{\setminus d}], t)\right)}{\sum_{s' \in \mathcal{X}} \exp\left(-f^\theta([s', \boldsymbol{x}^{\setminus d}], t)\right)}.$$

While EBMs offer remarkable modeling flexibility, they also come at a high computational cost. In particular, evaluating $\prod_{d=1}^D p_t^\theta(s|\boldsymbol{x}^{\setminus d})$ for all $s \in \mathcal{X}$ requires $O(D \times S)$ evaluations of the neural network f^θ, quickly becoming infeasible, even for moderate values of D and S. Furthermore, the study in [87] already demonstrated that alternative architectures could provide a more favorable balance between sample quality and computational efficiency. Therefore, in our experimental analysis, we explore these alternative models that promise to mitigate the computational burden associated with EBMs.

5.2. Masked Models

To address the computational intensity of EBMs without significantly compromising their flexibility, masked models emerge as an efficient solution [87]. Consider a masking function $m_d(\boldsymbol{x}) = [x^1, \ldots, x^{d-1}, \text{MASK}, x^{d+1}, \ldots, x^D]$ that replaces the d-th dimension of a given vector $\boldsymbol{x}$ with a special mask token MASK. This masking allows us to reparameterize the conditional distribution as

$$p_t^\theta(X_t^d | \boldsymbol{x}^{\setminus d}) = \text{softmax}\left(f^\theta(m_d(\boldsymbol{x}), t)\right),$$

where $f^\theta(\boldsymbol{x}, t) : \{\mathcal{X} \cup \text{MASK}\}^D \times \mathbb{R} \to \mathbb{R}^S$. In this setup, f^θ can be any general neural network, with the only additional requirement being the ability to handle the masked token. The masked model parameterization significantly reduces the computational complexity to $O(D)$ evaluations of f^θ, a substantial improvement over EBMs [87].

5.3. Hollow Transformer

While the previously discussed approaches offer considerable flexibility in neural network design, they inherently scale with the dimensionality. In response to this limitation, a novel variant of the Transformer architecture termed the hollow Transformer, has been proposed in [87]. The hollow Transformer significantly reduces the computational complexity to just $O(1)$ neural network evaluations. The key design feature of this model is to ensure that the diagonals of the Jacobian matrix of $p_t^\theta(\boldsymbol{X}_t|\boldsymbol{x})$ are zero for any given input $\boldsymbol{x}$ [87]. To achieve this property, several techniques have been employed in the past, such as autoregressive masking [23, 90], which results in a triangular Jacobian, and hollow masking [9], where full context is considered, but the interaction between dimensions is limited to a single dense layer.

As depicted in Figure 5.1, the hollow Transformer innovatively captures the full context for each dimension by running two autoregressive Transformers [90], one in each direction, across l_E layers. The resulting hollow Jacobian matrix is formed by summing upper and lower triangular Jacobians in an additional decoder Transformer block with l_D layers. This addition ensures that only the diagonals of the Jacobian matrix of $p_t^\theta(\boldsymbol{X}_t|\boldsymbol{x})$ are zero and the full context for each position is considered [87]. The unidirectional Transformers in this model are adaptations of the encoder part of the original Transformer architecture [90], tailored to meet the specific requirements of our application. Figure 5.2 shows a schematic illustration of the unidirectional Transformer. Each of the l_E unidirectional Transformer blocks consists of two main components: a multi-head self-attention [90] block using scaled dot-product attention [90] and a feedforward block.

Self-attention allows the model to dynamically assign importance to different parts of the input, making it powerful to capture dependencies and relationships within the input sequence [90]. The self-attention mechanism initiates by projecting the input data $\boldsymbol{x}$ into query Q, key K, and value V matrices, using learned weight matrices W^Q, W^K, and W^V, respectively. Then, the mechanism computes attention scores from the dot products QK^T, which indicate the relevance of elements within the sequence, and scales them by the square root of the dimension

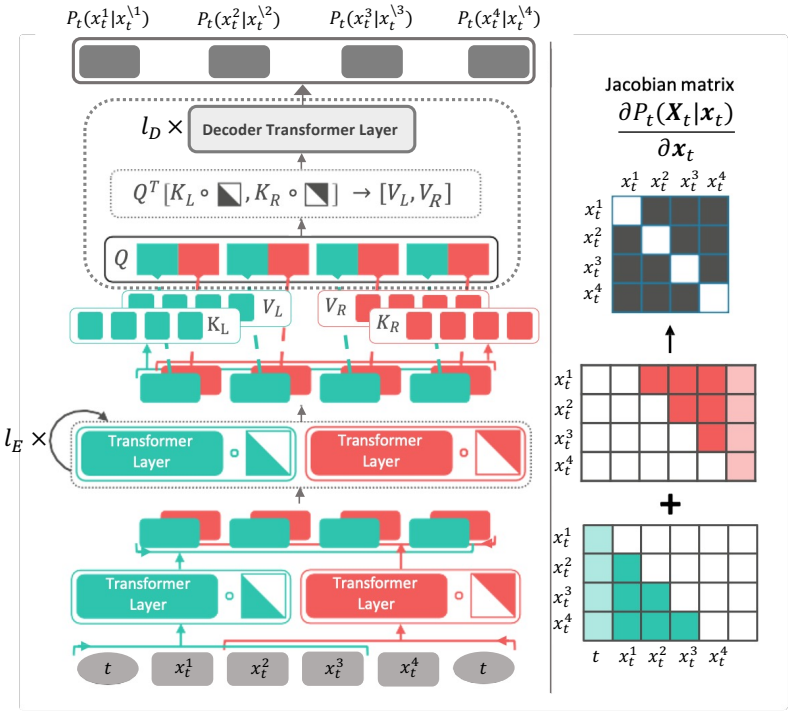


Figure 5.1.: Schematic illustration of the hollow Transformer architecture, based on [87].

of the key matrix $\sqrt{d_k}$ for stabilization. Subsequently, a masking operation is applied to either the lower or upper triangular part of the resulting attention score matrix, depending on the directionality of the unidirectional Transformer. This masking step leads to lower or upper triangular Jacobians of $p_t^\theta(\boldsymbol{X}_t|\boldsymbol{x})$. After the masking operation, a softmax function is applied to the attention scores matrix to derive attention weights. Elements with higher attention weights are considered more important and contribute more to the model's output [90]. The core mathematical formula for scaled dot-product attention can be expressed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V.$$

To further enhance the model's capacity to capture diverse patterns and relationships within the data, the unidirectional Transformers employ multiple attention heads [90]. These heads parallelize self-attention operations on the input data. Each attention head independently computes attention weights and applies them to the input data, allowing the model to capture different aspects simultaneously [90]. As depicted below, the attention weights in each head are computed using the same scaled dot-product mechanism:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_H)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left(\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_{k_i}}}\right)VW_i^V.$$

Here, H is the number of attention heads, and W_i^Q, W_i^K, and W_i^V are weight matrices for each head. The matrix W^O is an additional weight matrix used to concatenate the outputs from each head, and d_{k_i} is the dimension of the key in head i.

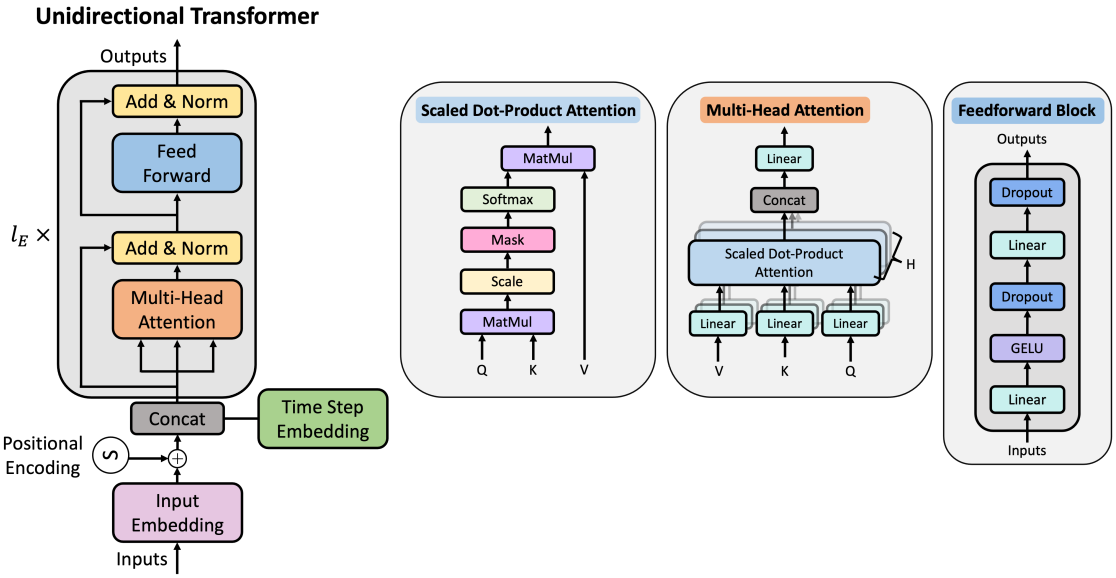


Figure 5.2.: Schematic illustration of the unidirectional Transformer with its core components: Multi-head self-attention using scaled dot-product attention and the feedforward layer, based on [90].

Following the attention block, each unidirectional Transformer block contains a position-wise feedforward network, as shown in Figure 5.2 on the right. It includes a GELU nonlinearity [32] and a dropout layer [35] positioned between two linear transformations and an additional dropout layer as the final output layer.

After each attention and feedforward block, the unidirectional Transformer incorporates layer normalizations [5] and residual connections [31] around each of these blocks. This design choice facilitates effective training and helps avoid problems associated with exploding or vanishing gradients in deep networks.

Prior to feeding the data into the first unidirectional Transformer block, the input data is transformed into a higher dimension d_{model} by a learned embedding. Similarly, the timestep is also embedded into this higher dimensional space. The timestep representation is then concatenated with the input data, enriching the input features with temporal context before being fed into the Transformer architecture. In addition, positional encodings are also used to inject sequence order information to compensate for the Transformer's inherent lack of sequential awareness [90]. This is done using sine and cosine functions of different frequencies, as in the standard Transformer architecture:

$$PE_{pos,2d} = \sin\left(\frac{pos}{10000^{\frac{2d}{d_{\text{model}}}}}\right)$$
$$PE_{pos,2d+1} = \cos\left(\frac{pos}{10000^{\frac{2d}{d_{\text{model}}}}}\right),$$

where pos is the position in the sequence.

At the top of the hollow Transformer is an additional Transformer block, illustrated in Figure 5.3. This Transformer essentially serves as a decoder and closely resembles the unidirectional Transformer depicted in Figure 5.2.

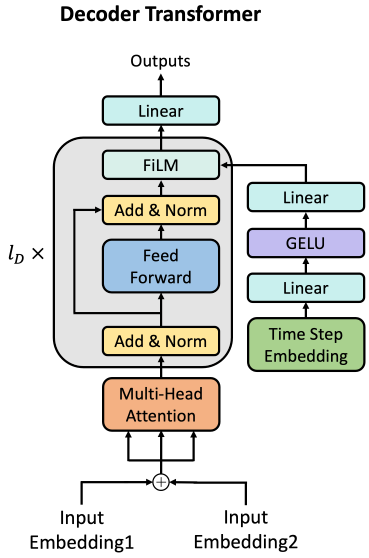


Figure 5.3.: Schematic illustration of the decoder Transformer block at the top of the hollow Transformer architecture.

However, in this context, the query matrix is obtained by summing the output embeddings of the two unidirectional Transformers. Attention is then simultaneously applied to the key and value matrices derived from these embeddings, ensuring that the diagonal elements of the Jacobian matrix of $p_t^\theta(\boldsymbol{X}_t|\boldsymbol{x})$ are zero. Additionally, this Transformer block integrates a Feature-wise Linear Modulation (FiLM) layer [71] as the final output layer, followed by a linear layer. The FiLM layer modulates intermediate activations based on time step embedding, which is first processed through a GELU activation function positioned between two linear layers. It scales and shifts feature maps at each step using learnable parameters, allowing the model to effectively capture temporal dependencies within sequential data [71].

In summary, the hollow Transformer maintains the core principles of the original architecture, while the modifications specifically address the challenges of dependency on dimensionality and state space size in terms of neural network evaluations. It effectively leverages the expressiveness of the well-established Transformer architecture, yet innovates to reduce computational demands in modeling the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$.

6. Simulating the Forward and Reverse Process

In our experiments, we aim to model data with D dimensions, where each dimension can take one of S possible values. In this setting, calculating transition probabilities for every possible subsequent state in the forward and reverse process simulation would require computing S^D rate values. This computation quickly becomes infeasible with growing values of S and D [7]. Consequently, this necessitates the exploration of alternative approaches for simulating the diffusion processes to enable efficient data generation in our experiments.

In light of these computational challenges, this chapter presents methodologies designed to simulate the forward and reverse process of diffusion models more efficiently. We begin by introducing the common dimension factorization approach for tractably computing the transition probabilities. We then outline the design choice of R_t to enable efficient simulation of the forward process and to ensure convergence of $q_T(\boldsymbol{x}_T)$ to $p_{\text{ref}}(\boldsymbol{x}_T)$. Subsequently, we present the simulation methods used in τLDR and SDDM to efficiently simulate the reverse process for generating samples. These methods are further adapted to accommodate the distinct parameterizations within each other's framework. This extension enables the use of a broader range of sampling strategies, as initially employed in their respective studies. In addition, we propose midpoint tau-leaping as an alternative simulation method to improve sample quality. In our experimental evaluation, we employ these various sampling methods to determine their efficacy in generating different discrete data types.

6.1. Simulation and Design of the Forward Process

To overcome the computational challenges of simulating the forward process $\{\boldsymbol{X}_t\}_{t\in[0,T]}$, we follow standard practice [3, 7] and factorize it, i.e., we decompose $\boldsymbol{X}_t = (X_t^1, \ldots, X_t^D)$ into its dimensions. The factorization enables each dimension's forward process $\{X_t^d\}_{t\in[0,T]}$ to propagate independently. Factorizing the forward process is a valid approach since the likelihood of simultaneous transitions in two or more dimensions is effectively zero for a continuous-time process [7, 87]. Therefore, any transition in the forward process involves a change in only one dimension at a time. For a system with S^D potential rate values, only $D \times (S-1) + 1$ values are nonzero, drastically reducing the number of necessary computations. The exact expression of the forward rates, for a factorized forward process $q_{t|r}(\boldsymbol{x}_t|\boldsymbol{x}_r) = \prod_{d=1}^D q_{t|r}(x_t^d|x_r^d),\ t > r$ [7], can then be expressed as (see proof in Appendix A.7 [7])

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

This factorization allows us to compute $q_{t|0}$ analytically in a tractable manner, thereby facilitating efficient training, as already mentioned in Chapter 4. We can compute $q_{t|0}$ analytically by solving the ODE in Equation 4.1 for each subprocess [7]. An analytical solution of this ODE is feasible if $R_t^d R_{t'}^d = R_{t'}^d R_t^d$ holds for all $t, t' \in [0, T]$ [74]. A simple approach to meet this condition is to establish $R_t^d = \beta(t) R_b$, where $R_b = U \Lambda U^{-1} \in \mathbb{R}^{S \times S}$ is a time-independent base rate matrix in its eigendecomposition, and $\beta(t) \in \mathbb{R}$ is a time-dependent noise schedule [87]. The subsequent analytical expression is as follows (see proof in Appendix A.6 [7])

$$q_{t|0}(x_t^d | x_0^d) = \left(U \exp\left(\Lambda \int_0^t \beta(v) \mathrm{d}v \right) U^{-1} \right).$$

For the convergence of $q_T(\boldsymbol{x}_T)$ to $p_{\text{ref}}(\boldsymbol{x}_T)$, we assume a factorization $p_{\text{ref}}(\boldsymbol{x}) = \prod_{d=1}^D p_{\text{ref}}(x^d)$. Then to achieve this convergence, it is sufficient to select a base matrix R_b that has $p_{\text{ref}}(x^d)$ for all $x^d \in \mathcal{X}$ as its stationary distribution [7]. To find such a R_b, it must satisfy the following relation of the differential equation of the marginals (refer to Equation 2.8)

$$\partial_t p_{\text{ref}}(x^d) = \sum_{z^d \in \mathcal{X}} p_{\text{ref}}(z^d) R_b(z^d, x^d) = 0, \qquad \forall x^d \in \mathcal{X}. \tag{6.1}$$

Note that if Equation 6.1 holds for a specific base rate matrix R_b, then $R_t^d = R_b \beta(t)$ will also have the factorized p_{ref} as its stationary distribution, since the multiplication by $\beta(t)$ can be seen as just a scaling of the time axis [7]. In our experimental evaluation, we consider different base rate matrices and noise schedules. For example, one can choose a linear noise schedule $\beta(t) = t$, along with a uniform base rate matrix $R_b = \mathbb{1}\mathbb{1}^T - S\mathbb{I}$, where $\mathbb{1}\mathbb{1}^T$ is a matrix of ones and $\mathbb{I}$ is the identity matrix. The corresponding stationary distribution p_{ref} is then the uniform distribution over all $s \in \mathcal{X}$ in each dimension [7].

6.2. Simulation of the Generative Reverse Process

We generate samples by simulating the time-reversed CTMC $\{\hat{\boldsymbol{X}}_t\}_{t \in [0,T]}$ with rate $\hat{R}_t^\theta$ from the reference distribution $p_{\text{ref}}(\boldsymbol{x}_T)$ at time $t = T$ back to time $t = 0$ towards the initial distribution $\pi_{\text{data}}(\boldsymbol{x}_0)$. Similar to the forward process, we factorize the reverse process to simulate this CTMC efficiently. Consequently, for the time-reversed CTMC, each transition also involves a change in exactly one dimension [7]. The factorized form of the reverse rate can be obtained by substituting the factorized forward rate into the expression for the reverse rates from Equation 4.2:

$$\hat{R}_t(\boldsymbol{x}, \boldsymbol{y}) = \sum_{d=1}^D R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{\setminus d}, \boldsymbol{y}^{\setminus d}} \frac{q_t(y^d, \boldsymbol{x}^{\setminus d})}{q_t(x^d, \boldsymbol{x}^{\setminus d})}.$$

For the parameterization as in τLDR, this results in

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

For the conditional marginal parameterization, as in SDDM, we have

$$\hat{R}_t^\theta(\boldsymbol{x}, \boldsymbol{y}) = \sum_{d=1}^D R_t^d(y^d, x^d) \delta_{\boldsymbol{x}^{\setminus d}, \boldsymbol{y}^{\setminus d}} \frac{p_t^\theta(y^d | \boldsymbol{x}^{\setminus d})}{p_t^\theta(x^d | \boldsymbol{x}^{\setminus d})}.$$

When using the factorized forms of R_t and $\hat{R}_t^\theta$, the loss functions L_{CRM}, L_{ll}, L_{ll^*} are still applicable, since they do not depend on these rates. For a factorized form of L_{eCTE}, refer to [7].

The factorized reverse process can be exactly simulated with the modified next reaction method [1] by first sampling the time until the process transitions to the next state and then sampling the next state from a categorical distribution based on the reverse rates [7]. However, we cannot simulate each dimension in parallel using this method [7, 87]. The reverse rates in a single dimension $\hat{R}_t^{\theta,d}$ are dependent on the values of $\boldsymbol{x}$ across all dimensions. This dependency necessitates the processing of each transition individually for an exact simulation [7]. Consequently, this leads to a significant increase in computational demand, making an exact simulation impractical, particularly when dealing with high-dimensional data.

Due to the computational complexity associated with an exact simulation of the reverse process, the τLDR and SDDM frameworks utilize various approximate simulation techniques as practical alternatives. In the following subsections, we introduce these simulation methods and adapt them to fit the specific parameterizations used in each framework. This adaptation enables the simulation methods to be applied interchangeably within the τLDR and SDDM frameworks. Additionally, we propose midpoint tau-leaping as an alternative simulation method to potentially improve the quality of generated samples.

To clarify the temporal aspects of the reverse process simulation, we will henceforth denote data $\boldsymbol{x}, \boldsymbol{y}$ with time subscripts $\boldsymbol{x}_t, \boldsymbol{y}_t$ to explicitly indicate the time point under consideration.

6.2.1. Euler Sampling

One approach used in SDDM for simulating each dimension in parallel in the reverse process is Euler sampling. Euler sampling diverges from the traditional step-by-step backward traversal in time. Instead, it implements larger time leaps, moving from a time point t to an earlier time $t-\tau$, where τ represents our time step, typically a small, non-negative increment. Within the time interval $[t, t-\tau]$, Euler's method assumes constant rates in the rate matrix $\hat{R}_t^{\theta,d}$ and a fixed state x_t^d for each dimension d. This allows the computation of reverse transition probabilities for each dimension at time $t-\tau$ as follows

$$p^\theta_{t-\tau|t}(X^d_{t-\tau} = s|\boldsymbol{x}_t) = \begin{cases} \tau \hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s), & \text{for } s \neq x_t^d \\ 1 - \tau \sum_{s' \neq x_t^d} \hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s'), & \text{for } s = x_t^d, \end{cases} \tag{6.2}$$

where the probabilities are normalized and non-negative probabilities are ensured by clipping [87]. The new values for each dimension are then sampled from these approximated transition probabilities:

$$x^d_{t-\tau} \sim p^\theta_{t-\tau|t}(X^d_{t-\tau}|\boldsymbol{x}_t).$$

In this way, the procedure allows us to simultaneously update all dimensions of $\boldsymbol{x}_t$, significantly improving the efficiency of simulating the reverse process in high-dimensional data.

Euler sampling can also be extended to fit the parameterization of τLDR. The transition probabilities of each dimension depend solely on τ and the reverse rates $\hat{R}_t^{\theta,d}$. Therefore, this method does not depend on the specific parameterization of the conditional marginals used to leverage the categorical ratio matching loss in SDDM. Consequently, we can also use the parameterization of $p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}_t)$ in τLDR to first approximate the reverse rates as follows

$$\hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s) = R_t^d(s, x_t^d) \sum_{x_0^d} p_{0|t}^{\theta}(x_0^d|\boldsymbol{x}_t) \frac{q_{t|0}(s|x_0^d)}{q_{t|0}(x_t^d|x_0^d)}.$$

Subsequently, we can calculate the transition probabilities similarly to Equation 6.2 and update $\boldsymbol{x}_t$ by sampling the new states from these transition probabilities. This adaptability enables the integration of Euler sampling within the τLDR framework, thus expanding the range of applicable sampling strategies.

The complete process of generating samples using Euler sampling is described in Pseudo-algorithm 3. Initially, it involves sampling $\boldsymbol{x}_t$ from $p_{\text{ref}}(\boldsymbol{x}_T)$ at time $t = T$ and computing the reverse rates with $\boldsymbol{x}_t$ based on the specific parameterization of the respective frameworks. Following this, we compute the transition probabilities and sample the update $\boldsymbol{x}_{t-\tau}$ from these transition probabilities. Subsequently, t is decremented by τ, and the process iterates until t reaches 0, and we obtain the generated samples.

6.2.2. Tau-Leaping

Alternatively, the τLDR framework employs tau-leaping [24] to simulate the reverse process efficiently. Similar to Euler sampling, tau-leaping involves making larger time steps from a time point t to an earlier time $t - \tau$ while assuming the constancy of $\hat{R}_t^{\theta}$ and $\boldsymbol{x}_t$ during the leap. However, unlike Euler sampling, tau-leaping estimates the number of transitions occurring within the interval $[t, t-\tau]$ and applies them simultaneously at $t - \tau$, maintaining $\boldsymbol{x}_t$ constant throughout the interval. The number of transitions from $\boldsymbol{x}_t$ to a different state $\boldsymbol{y}$ within $[t, t-\tau]$ are modeled as a Poisson distribution with mean $\tau\hat{R}_t^{\theta}(\boldsymbol{x}_t, \boldsymbol{y}_i)$. The update mechanism is then given by

$$\boldsymbol{x}_{t-\tau} = \boldsymbol{x}_t + \sum_i P_i(\boldsymbol{y}_i - \boldsymbol{x}_t), \tag{6.3}$$

where P_i denotes a Poisson distributed random variable with the specified mean [7]. Note that $\sum_i P_i(\boldsymbol{y}_i - \boldsymbol{x}_t)$ assumes a mapping from the set of possible state values in each dimension $\mathcal{X}$ to $\mathbb{Z}$ [7]. Since the Poisson random variable can have arbitrarily large sample values, this mapping implies that jumps can lead to out-of-bound values. In such cases, these jumps are constrained within the bounds [7].

The update in Equation 6.3 can be further simplified by considering that $\hat{R}_t^{\theta}(\boldsymbol{x}_t, \boldsymbol{y})$ is only nonzero when $\boldsymbol{y}$ has a different value to $\boldsymbol{x}_t$ in exactly one dimension [7]. By explicitly summing over these options, the state at $t - \tau$ can be expressed as

$$\boldsymbol{x}_{t-\tau} = \boldsymbol{x}_t + \sum_{d=1}^{D} \sum_{s=1\setminus x_t^d}^{S} P_{ds}(s - x_t^d)\boldsymbol{e}_d, \tag{6.4}$$

Algorithm 3 Reverse Process Simulation with Euler Sampling

Require: T ▷ Hyperparameter: Starting time
Require: τ ▷ Hyperparameter: Step size
  $t \leftarrow T$
  $\boldsymbol{x}_t \sim p_{\mathrm{ref}}(\boldsymbol{x}_T)$
  while $t > 0$ **do**
    if τLDR framework **then**
      Compute $p^{\theta}_{0|t}(x^d_0|\boldsymbol{x}_t)$, for $d = 1, \ldots, D$
    else if SDDM framework **then**
      Compute $p^{\theta}_t(s|\boldsymbol{x}_t^{\backslash d})$, for $d = 1, \ldots, D$, for all $s \in \mathcal{X}$
    end if
    for $d = 1, \ldots, D$ **do**
      for every $s \in \mathcal{X}$ **do**
        if τLDR framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \sum_{x^d_0} p^{\theta}_{0|t}(x^d_0|\boldsymbol{x}_t) \frac{q_{t|0}(s|x^d_0)}{q_{t|0}(x^d_t|x^d_0)}$
        else if SDDM framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \frac{p^{\theta}_t(s|\boldsymbol{x}_t^{\backslash d})}{p^{\theta}_t(x^d_t|\boldsymbol{x}_t^{\backslash d})}$
        end if
        $p^{\theta}_{t-\tau|t}(X^d_{t-\tau} = s|\boldsymbol{x}_t) \leftarrow \begin{cases} \tau \hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s), & \text{for } s \neq x^d_t \\ 1 - \tau \sum_{s' \neq x^d_t} \hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s'), & \text{for } s = x^d_t \end{cases}$
      end for
    end for
    for $d = 1, \ldots, D$ **do**
      $x^d_{t-\tau} \sim p^{\theta}_{t-\tau|t}(X^d_{t-\tau}|\boldsymbol{x}_t)$
    end for
    $t \leftarrow t - \tau$
  end while

where $\boldsymbol{e}_d$ is a one-hot vector, and P_{ds} is a Poisson variable with a mean of $\tau \hat{R}^{\theta}_t(\boldsymbol{x}_t, \boldsymbol{x}_t + (s - x^d_t)\boldsymbol{e}_d)$ [7]. Since multiple P_{ds} can be nonzero, tau-leaping allows for changes in multiple dimensions in a single step. In addition, unlike Euler sampling, multiple transitions within the same dimension ($\sum_{s=1\backslash x^d_t}^{S} P_{ds} > 1$) are also possible.

To better illustrate the tau-leaping scheme, Figure 6.1 provides a 3D representation of a single tau-leaping step with 2-dimensional data $\boldsymbol{x}_t$ and five states. At time t, $\boldsymbol{x}_t$ is in state S_4 in the first dimension and in S_1 in the second dimension, making it $\boldsymbol{x}_t = \{S_4, S_1\}$. Then, during the interval $[t, t - \tau]$, the tau-leaping scheme estimates one jump in the first dimension to S_1 ($P_{11} = 1$), and three jumps in the second dimension, two from S_1 to S_2 ($P_{22} = 2$) and another from S_1 to S_3 ($P_{23} = 1$). In the case of ordinal data, these multiple jumps within the same dimension have meaningful interpretations since the mapping from $\mathcal{X}$ to $\mathbb{Z}$ is not arbitrary. Applying these jumps to $\boldsymbol{x}_t$ at time $t - \tau$, the resulting states are $\boldsymbol{x}_{t-\tau} = \{S_1, S_5\}$, as shown in Figure 6.1. However, for non-ordinal data, where the mapping from $\mathcal{X}$ to $\mathbb{Z}$ is arbitrary, multiple jumps within the same dimension are not meaningful [7]. In such cases, changes to x^d_t for which $\sum_{s \backslash x^d_t} P_{ds} > 1$ holds are rejected, following [7]. Thus, for non-ordinal data, the resulting states in the above example would be $\boldsymbol{x}_{t-\tau} = \{S_1, S_1\}$ since $P_{22} + P_{23} > 1$.

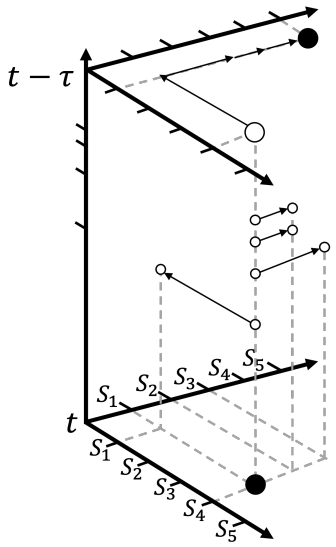


Figure 6.1.: Visualization of a tau-leaping step for two-dimensional data with five states, adapted from [7].

Tau-leaping can also be used with the conditional parameterization of the reverse rates, as required for training models with the categorical ratio matching loss. As discussed previously in the context of Euler sampling, where the transition probabilities in each dimension depend only on τ and the reverse rate $\hat{R}_t^{\theta,d}$, now the rates of the Poisson distribution depend solely on them. Thus, similar to our argument for Euler sampling, tau-leaping can be adapted to the SDDM framework. Therefore, by first calculating the reverse rates with the conditional parameterization as follows

$$\hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s) = R_t^d(s, x_t^d) \frac{p_t^\theta(s|\boldsymbol{x}_t^{\setminus d})}{p_t^\theta(x_t^d|\boldsymbol{x}_t^{\setminus d})},$$

we can sample from a Poisson distribution specified by $\hat{R}_t^{\theta,d}$ and then use the update scheme for tau-leaping, as shown in Equation 6.4.

The complete procedure for generating samples using tau-leaping, as depicted in Pseudo-algorithm 4, begins by sampling $\boldsymbol{x}_t$ at time $t = T$ from our reference distribution. We then compute the reverse rates for $\boldsymbol{x}_t$ according to our chosen framework. Subsequently, we sample the number of jumps P_{ds} to each state s in each dimension d from a Poisson distribution, determined by our reverse rates and the time step. Next, we update our current states in $\boldsymbol{x}_t$ by adding the product of the estimated number of jumps and the jump size $(s - x_t^d)$. Additionally, we handle out-of-bounds jumps by clipping them to ensure a valid $\boldsymbol{x}_{t-\tau}$. Afterward, we decrement t by τ and repeat this process until we reach $t = 0$, at which point we have our final samples.

Algorithm 4 Reverse Process Simulation with Tau-Leaping

Require: T ▷ Hyperparameter: Starting time
Require: τ ▷ Hyperparameter: Step size
  $t \leftarrow T$
  $\boldsymbol{x}_t \sim p_{\text{ref}}(\boldsymbol{x}_T)$
  while $t > 0$ **do**
    if τLDR framework **then**
      Compute $p^{\theta}_{0|t}(x^d_0|\boldsymbol{x}_t)$, for $d = 1, \ldots, D$
    else if SDDM framework **then**
      Compute $p^{\theta}_t(s|\boldsymbol{x}^{\backslash d}_t)$, for $d = 1, \ldots, D$, for $s = 1, \ldots, S$
    end if
    for $d = 1, \ldots, D$ **do**
      for $s = 1, \ldots, S \backslash x^d_t$ **do**
        if τLDR framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \sum_{x^d_0} p^{\theta}_{0|t}(x^d_0|\boldsymbol{x}_t) \frac{q_{t|0}(s|x^d_0)}{q_{t|0}(x^d_t|x^d_0)}$
        else if SDDM framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \frac{p^{\theta}_t(s|\boldsymbol{x}^{\backslash d}_t)}{p^{\theta}_t(x^d_t|\boldsymbol{x}^{\backslash d}_t)}$
        end if
        $P_{ds} \leftarrow \text{Poisson}\left(\tau \hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s)\right)$
      end for
    end for
    for $d = 1, \ldots, D$ **do**
      if data is non-ordinal AND $\sum_{s=1\backslash x^d_t}^{S} P_{ds} > 1$ **then**
        $x^d_{t-\tau} \leftarrow x^d_t$
      else
        $x^d_{t-\tau} \leftarrow x^d_t + \sum_{s=1\backslash x^d_t}^{S} P_{ds} \times (s - x^d_t)$
      end if
    end for
    $\boldsymbol{x}_{t-\tau} \leftarrow \text{Clip}\left(\boldsymbol{x}_{t-\tau}, \text{min} = 1, \text{max} = S\right)$
    $t \leftarrow t - \tau$
  end while

Both the tau-leaping and Euler sampling approximations improve with decreasing τ, converging to an exact simulation as τ approaches zero [7, 87]. In an exact simulation, changes are limited to one dimension at a time. By incrementally increasing τ in tau-leaping and the Euler scheme, we allow multiple dimensions to change in a single step. Such adaptability enables us to trade the simulation's fidelity and, thus, sample quality with the speed of the simulation [7]. In our experiments, we leverage this flexibility to find optimal tradeoffs between sample quality and simulation speed among the samplers. Based on our experimental findings, we provide recommendations for choosing the most effective sampling methods tailored to the specific characteristics of each data domain. Our goal is to equip future research with strategic insights that facilitate the rapid generation of samples without compromising sample quality.

6.2.3. Predictor-Corrector

Tau-leaping and Euler Sampling operate under the assumption that the rate matrix R_t remains constant during the interval $[t, t-\tau]$. While this facilitates efficient sampling, it can introduce deviations of $\boldsymbol{x}_t$ from the true marginal $q_t(\boldsymbol{x}_t)$, especially when larger time steps are involved [7]. To address this issue, a novel Predictor-Corrector scheme for discrete state spaces has been introduced in [7], aiming to align the marginal distribution of samples at time t more closely with $q_t(\boldsymbol{x}_t)$. The Predictor-Corrector scheme was initially combined with tau-leaping. However, it can also be used with Euler sampling since it only involves adjusting the reverse rates in the corrector step, as we will see in the following.

As depicted in Pseudo-algorithm 5, the Predictor-Corrector scheme starts with a predictor step, providing an initial approximation for the next states in the reverse process. For tau-leaping and Euler sampling, this involves leaping from t to $t-\tau$, assuming that $\hat{R}_t^\theta$ and $\boldsymbol{x}_t$ remain constant during this interval. Then, we sample the next states using $\hat{R}_t^\theta$, as described in Pseudo-algorithms 3 and 4. At a predetermined time point t_C during the reverse process simulation, we start applying a specified number of additional corrector steps N_C. In a corrector-step, we simulate the next states with the same method (e.g., tau-leaping or Euler), but using a corrected rate matrix $R_t^{c\theta} = \hat{R}_t^\theta + R_t$. The corrected rate R_t^c has $q_t(\boldsymbol{x}_t)$ as its stationary distribution (see proof in Appendix A.8 [7]). Thus, simulating the CTMC with this rate during the corrector step further aligns the samples to the true marginal $q_t(\boldsymbol{x}_t)$ [7]. By alternating between simulating the reverse CTMC with learned reverse rates (Predictor) and corrected rates (Corrector), the Predictor-Corrector scheme facilitates exploration within the domain of $q_t(\boldsymbol{x}_t)$. This dynamic approach can potentially improve sample quality and provides a flexible mechanism to balance computational efficiency with fidelity to the desired distribution [7]. The ability to adjust the timing and number of corrector steps allows for fine-tuning according to the specific requirements of the sampling process, enhancing the method's adaptability and effectiveness.

Algorithm 5 Reverse Process Simulation with the Predictor-Corrector Scheme

Require: T ▷ Hyperparameter: Starting time
Require: τ ▷ Hyperparameter: Step size
Require: t_C ▷ Hyperparameter: Time at which we start applying corrector steps
Require: N_C ▷ Hyperparameter: Number of corrector steps
  $t \leftarrow T$
  $\boldsymbol{x}_t \sim p_{\text{ref}}(\boldsymbol{x}_T)$
  while $t > 0$ **do**
    $\boldsymbol{x}_t \leftarrow \text{PredictorStep}(\boldsymbol{x}_t, t)$ ▷ "Normal" Euler/tau-leaping step with rate R_t^θ
    if $t \leq t_C$ **then**
      for $n = 1, \ldots, N_C$ **do**
        $\boldsymbol{x}_t \leftarrow \text{CorrectorStep}(\boldsymbol{x}_t, t)$ ▷ Euler/tau-leaping step with rate $R_t^{c\theta}$
      end for
    end if
    $t \leftarrow t - \tau$
  end while

In one of our experiments, we explore various configurations of different numbers of corrector steps. We aim to identify tradeoffs between applying additional corrector steps, which may enhance sample quality, and maintaining an appropriate simulation speed.

6.2.4. Analytical Sampling

While introducing corrector steps can improve sample quality, it also incurs additional computational costs. To circumvent this issue, a sampling method leveraging the implicit parameterization of the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$ has been proposed in [87]. As previously mentioned in Section 4.3.3, we can approximate $q_t(X_t^d|\boldsymbol{x}_t^{\setminus d})$ with $p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d})$ by efficiently computing

$$p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d}) = \sum_{x_0^d} p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|0}(X_t^d|x_0^d),$$

which enables the continued use of the categorical ratio matching loss in Equation 4.16. In addition, this implicit modeling of the conditional marginals allows to generate samples when considering the true reverse process

$$\begin{aligned} q_{t-\tau|t}(X_{t-\tau}^d|\boldsymbol{x}_t) &= \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}_t) q_{t-\tau|t,0}(X_{t-\tau}^d|\boldsymbol{x}_t, x_0^d) \\ &= \sum_{x_0^d} \frac{q(x_t^d|x_0^d, \boldsymbol{x}_t^{\setminus d}) q(x_0^d|\boldsymbol{x}_t^{\setminus d})}{q(x_t^d|\boldsymbol{x}_t^{\setminus d})} \frac{q(x_t|x_0^d, X_{t-\tau}^d) q(X_{t-\tau}^d|x_0^d)}{q(x_t|x_0^d)} \\ &\propto \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|t-\tau}(x_t^d|X_{t-\tau}^d) q_{t-\tau|0}(X_{t-\tau}^d|x_0^d). \end{aligned} \tag{6.5}$$

By substituting the parameterization $p_{0|t}^\theta(X_0^d|\boldsymbol{x}_t^{\setminus d})$ into Equation 6.5, resulting in

$$p_{t-\tau|t}^\theta(X_{t-\tau}^d|\boldsymbol{x}_t) \propto \sum_{x_0^d} p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|t-\tau}(x_t^d|X_{t-\tau}^d) q_{t-\tau|0}(X_{t-\tau}^d|x_0^d), \tag{6.6}$$

we yield an analytical expression of reverse process sampling, circumventing the simulation error typically associated with Euler sampling or tau-leaping [87]. Hence, this method is called analytical sampling [87].

The complete procedure for generating samples with analytical sampling is illustrated in Pseudo-algorithm 6. Initially, we sample $\boldsymbol{x}_t$ from $p_{\text{ref}}(\boldsymbol{x}_T)$ at time $t = T$. Following this, we compute $p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d})$ using a neural network and then calculate the reverse transition probabilities $p_{t-\tau|t}^\theta$ by multiplying it with the forward transition probabilities $q_{t|t-\tau}$ and $q_{t-\tau|0}$, as expressed in Equation 6.6. Subsequently, we sample $\boldsymbol{x}_{t-\tau}$ from these approximated reverse transition probabilities. Finally, we decrement t by τ and repeat this procedure until we reach $t = 0$ and obtain our generated samples.

Algorithm 6 Reverse Process Simulation with Analytical Sampling

Require: T ▷ Hyperparameter: Starting time
Require: τ ▷ Hyperparameter: Step size
  $t \leftarrow T$
  $\boldsymbol{x}_t \sim p_{\text{ref}}(x_T)$
  while $t > 0$ **do**
    Compute $p_{0|t}(x_0^d | \boldsymbol{x}_t^{\setminus d})$ for $d = 1, \ldots, D$
    for $d = 1, \ldots, D$ **do**
      for every $s \in \mathcal{X}$ **do**
        Compute $p^\theta_{t-\tau|t}(X^d_{t-\tau} = s | \boldsymbol{x}_t) \propto \sum_{x_0^d} p^\theta_{0|t}(x_0^d | \boldsymbol{x}_t^{\setminus d}) q_{t|t-\tau}(x_t^d | s) q_{t-\tau|0}(s | x_0^d)$
      end for
    end for
    for $d = 1, \ldots, D$ **do**
      $x^d_{t-\tau} \sim p^\theta_{t-\tau|t}(X^d_{t-\tau} | \boldsymbol{x}_t)$
    end for
    $t \leftarrow t - \tau$
  end while

In [87], a small study was conducted to compare analytical sampling with Euler sampling. The findings revealed that analytical sampling could be simulated with significantly larger time steps to achieve reasonable sample quality compared to Euler sampling. Consequently, it was concluded that analytical sampling may offer greater robustness [87]. In our experimental analysis, we similarly aim to assess the robustness of analytical sampling and include it in our study of finding optimal tradeoffs between simulation speed and sample quality among the samplers.

However, it is important to note that this simulation method utilizes the conditional parameterization required for the categorical ratio matching loss. Unfortunately, we could not devise an equivalent scheme with the parameterization used in τLDR. Therefore, in our experimental analysis, we can solely apply this scheme within the SDDM framework.

6.2.5. Midpoint Tau-Leaping

While all previous methods require only one neural network evaluation per sampling step, we introduce midpoint tau-leaping [25] as a higher-order method to potentially improve the quality of generated samples. To the best of our knowledge, this method has not been used before in diffusion models. It is based on the midpoint Runge-Kutta for ODEs and has been proposed in the context of chemical physics to improve the convergence and stability properties of tau-leaping methods [25]. As illustrated in Pseudo-algorithm 7, it works by first predicting the states at the midpoint $t - \frac{\tau}{2}$ with

$$x^d_{t-\frac{\tau}{2}} = x^d_t + \left\lfloor \frac{1}{2}\tau \sum_{s=1 \setminus x_t^d}^{S} \hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s)(s - x_t^d) \right\rfloor,$$

where $\lfloor \cdot \rceil$ denotes rounding to the next integer. Subsequently, with these intermediate states $\boldsymbol{x}_{t-\frac{\tau}{2}}$, we compute new reverse rates $\hat{R}_t^{\theta,d}(\boldsymbol{x}_{t-\frac{\tau}{2}}, s)$. Then we sample the number of transitions for each $s \in \mathcal{X}$, $s \neq x^d_{t-\frac{\tau}{2}}$, in each dimension d from a Poisson random variable $P_{ds} = \text{Poisson}\left(\tau \hat{R}_t^{\theta,d}(\boldsymbol{x}_{t-\frac{\tau}{2}}, s)\right)$ and update our states by

$$x^d_{t-\tau} = x^d_t + \sum_{s=1 \setminus x^d_t}^{S} P_{ds} \times (s - x^d_t).$$

Apart from this additional step in the update scheme, the remainder of the algorithm follows the "standard" tau-leaping scheme. Similar to our argument for tau-leaping, it is evident from our update rule that the update of $\boldsymbol{x}_t$ relies solely on the reverse rates. Consequently, it is compatible with the parameterizations used in both the τLDR and SDDM frameworks.

We integrate midpoint tau-leaping into one of our experiments to evaluate its impact on performance, specifically examining if it provides a better balance between sample quality and simulation speed.

Algorithm 7 Reverse Process Simulation with Midpoint Tau-Leaping

Require: T ▷ Hyperparameter: Starting time
Require: τ ▷ Hyperparameter: Step size
  $t \leftarrow T$
  $\boldsymbol{x}_t \sim p_{\text{ref}}(\boldsymbol{x}_T)$
  while $t > 0$ **do**
    if τLDR framework **then**
      Compute $p^{\theta}_{0|t}(x_0^d|\boldsymbol{x}_t)$, for $d = 1, \ldots, D$
    else if SDDM framework **then**
      Compute $p^{\theta}_t(s|\boldsymbol{x}_t^{\backslash d})$, for $d = 1, \ldots, D$, for $s = 1, \ldots, S$
    end if
    for $d = 1, \ldots, D$ **do**
      for $s = 1, \ldots, S \backslash x_t^d$ **do**
        if τLDR framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \sum_{x_0^d} p^{\theta}_{0|t}(x_0^d|\boldsymbol{x}_t) \frac{q_{t|0}(s|x_0^d)}{q_{t|0}(x_t^d|x_0^d)}$
        else if SDDM framework **then**
          $\hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s) \leftarrow R^d_t(s, x^d_t) \frac{p^{\theta}_t(s|\boldsymbol{x}_t^{\backslash d})}{p^{\theta}_t(x_t^d|\boldsymbol{x}_t^{\backslash d})}$
        end if
      end for
      $x^d_{t-\frac{\tau}{2}} \leftarrow x^d_t + \left\lfloor \frac{1}{2}\tau \sum_{s=1}^{S} \hat{R}^{\theta,d}_t(\boldsymbol{x}_t, s)(s - x^d_t) \right\rceil$
    end for
    $\boldsymbol{x}_{t-\frac{\tau}{2}} \leftarrow \text{Clip}\left(\boldsymbol{x}_{t-\frac{\tau}{2}}, \text{min} = 1, \text{max} = S\right)$
    if τLDR framework **then**
      Compute $p^{\theta}_{0|t-\frac{\tau}{2}}(x_0^d|\boldsymbol{x}_{t-\frac{\tau}{2}})$, for $d = 1, \ldots, D$
    else if SDDM framework **then**
      Compute $p^{\theta}_{t-\frac{\tau}{2}}(X^d_{t-\frac{\tau}{2}}|\boldsymbol{x}^{\backslash d}_{t-\frac{\tau}{2}})$, for $d = 1, \ldots, D$
    end if
    for $d = 1, \ldots, D$ **do**
      for $s = 1, \ldots, S \backslash x^d_{t-\frac{\tau}{2}}$ **do**
        if τLDR framework **then**
          $\hat{R}^{\theta,d}_{t-\frac{\tau}{2}}(\boldsymbol{x}_{t-\frac{\tau}{2}}, s) \leftarrow R^d_{t-\frac{\tau}{2}}(s, x^d_{t-\frac{\tau}{2}}) \sum_{x_0^d} p^{\theta}_{0|t-\frac{\tau}{2}}(x_0^d|\boldsymbol{x}_{t-\frac{\tau}{2}}) \frac{q_{t-\frac{\tau}{2}|0}(s|x_0^d)}{q_{t-\frac{\tau}{2}|0}(x^d_{t-\frac{\tau}{2}}|x_0^d)}$
        else if SDDM framework **then**
          $\hat{R}^{\theta,d}_{t-\frac{\tau}{2}}(\boldsymbol{x}_{t-\frac{\tau}{2}}, s) \leftarrow R^d_{t-\frac{\tau}{2}}(s, x^d_{t-\frac{\tau}{2}}) \frac{p^{\theta}_{t-\frac{\tau}{2}}(s|\boldsymbol{x}^{\backslash d}_{t-\frac{\tau}{2}})}{p^{\theta}_t(x^d_{t-\frac{\tau}{2}}|\boldsymbol{x}^{\backslash d}_{t-\frac{\tau}{2}})}$
        end if
        $P_{ds} \leftarrow \text{Poisson}\left(\tau \hat{R}^{\theta,d}_{t-\frac{\tau}{2}}(\boldsymbol{x}_{t-\frac{\tau}{2}}, s)\right)$
      end for
    end for
    for $d = 1, \ldots, D$ **do**
      if data is non-ordinal AND $\sum_{s=1\backslash x^d_{t-\frac{\tau}{2}}}^{S} P_{ds} > 1$ **then**
        $x^d_{t-\tau} \leftarrow x^d_t$
      else
        $x^d_{t-\tau} \leftarrow x^d_t + \sum_{s=1\backslash x^d_t}^{S} P_{ds} \times (s - x^d_t)$
      end if
    end for
    $\boldsymbol{x}_{t-\tau} \leftarrow \text{Clip}\left(\boldsymbol{x}_{t-\tau}, \text{min} = 1, \text{max} = S\right)$
    $t \leftarrow t - \tau$
  end while

7. Experimental Setup

In this chapter, we define the experimental setup that forms the basis of our experimental analysis in the following chapter. We begin by setting our experimental agenda, i.e., outlining the specific aspects and questions we intend to investigate. Next, we describe the common implementation details we apply to all experiments and datasets to ensure methodological consistency. Finally, we introduce the selected datasets for our investigation and their respective training and evaluation setups. These datasets range from a low-dimensional binary dataset to a moderately higher-dimensional categorical dataset, culminating with a high-dimensional ordinal dataset with many state spaces. Each dataset presents different challenges and opportunities to yield profound insights.

7.1. Experimental Procedure

To address computational constraints while ensuring methodological consistency, our evaluation unfolds in stages, focusing on different aspects of the applied methodologies. Due to computational complexity considerations, we limit the use of midpoint tau-leaping and the Predictor-Corrector scheme to the ordinal dataset. Our structured evaluation plan, which is systematically applied to each dataset, proceeds as follows:

Implicit vs. Explicit Parameterization: We begin our analysis by comparing the sample quality of models using the implicit parameterization of the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d}) = \sum_{x_0^d} p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|0}(X_t^d|x_0^d)$ with models that explicitly predict $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$ within the SDDM framework. Given our computational constraints, we only train these models with L_{CRM}. This comparison aims to identify which parameterization of the conditional marginals offers superior performance in terms of sample quality. For simplicity, we refer to models that implicitly parameterize the conditional marginals as "implicit models" and models that directly predict $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$ as "explicit models".

Loss Function Evaluation and Cross-Framework Comparison: After our initial analysis of parameterization methods within SDDM, we extend our investigation by training two configurations of the previously analyzed models with alternative loss functions. Specifically, we use only the parameterization approach that led to the highest sample quality in the previous analysis. One model configuration is trained using the L_{CTEcrm} loss objective, while the other uses the L_{CRMll} loss. For the τLDR framework, we adopt a similar approach by training models using distinct loss functions: one with L_{CTEll} and another with L_{ll}. This experiment aims to determine whether training models with our proposed loss functions — L_{CTEcrm} and

L_{CRMll} in SDDM, and L_{ll} in τLDR — can improve sample quality compared to training models with the initially employed loss functions — L_{CRM} in SDDM, and L_{CTEll} in τLDR.

In addition, we compare the sample quality between the τLDR and SDDM frameworks and train the discrete-time discrete state space model (D3PM framework) [3] and report its results as a baseline. The discrete-time model corresponds to the discrete-time formulation of diffusion models outlined in Chapter 3, but now, we specifically consider a D-dimensional discrete state space $\mathcal{X}^D$. Consequently, similar to the continuous-time frameworks, we assume a factorization of the forward and reverse processes over the dimensions. To ensure fair comparisons across these frameworks, we adjust the discrete-time model to use a number of time steps during training that align its number of function evaluations (NFEs), i.e., neural network evaluations, during sampling with those of the continuous-time models. This comparative study seeks to determine whether the continuous-time frameworks outperform the discrete-time framework in terms of sample quality and to examine which continuous-time framework is better suited for generating specific types of discrete data. For simplicity, we refer to the models trained within the τLDR framework as τLDR models and those trained within the discrete-time discrete state space framework as D3PM models or discrete-time models.

Efficiency Comparison: Previous work in [87] has highlighted the advantages of masked models and hollow Transformers in achieving comparable sample quality to EBMs but with significantly reduced training and sampling times. Motivated by these findings, we extend the analysis to assess not only the sample quality achieved by these frameworks, but also their training and sampling efficiency. This analysis involves comparing training time, sampling time, and the maximum number of generated samples before encountering memory constraints. Our goal is to identify optimal tradeoffs between sample quality and computational efficiency among the different models and frameworks.

Sampling Study: In this experiment, we evaluate the performance of different sampling methods while varying the time step τ. We aim to determine the effectiveness of each sampler in maintaining sample quality as the time step increases, revealing optimal tradeoffs between sample quality and simulation speed among the samplers. We examine the performance of the sampling procedures using a single model selected from our earlier evaluations to maintain computational efficiency.

By following this structured evaluation plan, we aim to make meaningful comparisons between the different modeling techniques, allowing us to provide recommendations on selecting suitable modeling approaches for generating different types of discrete data. Through this guidance, we intend to equip future researchers with a set of strategies for effectively modeling discrete data within continuous-time models, thereby laying the groundwork for further advancements in future research in this field.

7.2. Common Implementation Details

In our experimental setup, we establish several common implementation details that are consistent across all experiments and datasets to ensure stability and uniformity.

When evaluating the loss functions on each minibatch of training data, we need to sample a time t from a uniform distribution $\mathcal{U}(0, T)$. However, setting t very close to zero can lead to training instabilities due to ill-conditioned reverse rates $\hat{R}_t$ in this region [7]. This phenomenon occurs because the marginal probability $q_t(\boldsymbol{x}_t)$ becomes highly peaked around the data manifold, causing $\log q_t(\boldsymbol{x}_t)$ to spike in regions far from the data [7]. To address this issue, we employ a common strategy: setting a minimum time ϵ such that t follows $\mathcal{U}(\epsilon, T)$. Following [7], we set ϵ to $0.01T$, where $T = 1$. This ensures minimal noise at $t = \epsilon$ and produces samples close to $\pi_{\text{data}}(\boldsymbol{x}_0)$ when simulating the reverse process from $t = T$ to $t = \epsilon$ [7].

We adopt forward rates of the form $R_t^d = \beta(t)R_b$ to simulate the forward process. This approach enables the analytical computation of $q_{t|0}$ and facilitates parallel simulation of all dimensions, thereby enhancing computational efficiency [7].

Throughout training, we maintain an exponential moving average of the model parameters with a decay factor of 0.9999. We use these averaged parameters to generate samples [7, 87].

To ensure a fair comparison between the frameworks, the D3PM model employs the same architecture and diffusion scheme as τLDR. In the SDDM framework, we also use the same diffusion scheme and a similar neural network architecture for the masked model parameterization. We configure the hollow Transformer in SDDM to have the same number of parameters as the other neural networks. Hyperparameters are selected by following previous approaches [3, 7, 87] and performing a grid search.

To maintain consistency within each dataset, all experiments conducted on the same dataset utilize the same training setup. Furthermore, for all experiments on the same dataset that focus on evaluating sample quality, the evaluation setup remains consistent.

We implemented the SDDM framework in PyTorch [70] for our experiments. For τLDR and D3PM, we used the existing PyTorch implementation and built our specifications and models upon this. We conduct training and sampling on a single NVIDIA RTX A6000 GPU with 48GB of VRAM.

7.3. Setup Spiral Dataset

For our binary dataset, we use the synthetic spiral dataset, following a setup similar to [87]. To create this dataset, we initially generate 2D continuous data by sampling from the unknown data distribution [14]. Next, we convert each data dimension into a 16-bit Gray code [28]. This conversion results in a binary dataset with discrete dimensions of $D = 32$ and a state size of

$S = 2$. Figure 7.1 illustrates the spiral dataset, showing a subset of 10,000 data samples.

Figure 7.1.: Spiral data samples.

7.3.1. Training Setup for the Spiral Dataset

To parameterize the masked model, the τLDR model, and the D3PM model, we use a BERT-based architecture [45], as illustrated in Figure 7.2. This architecture closely resembles the network structure used in the image modeling task in [87]. The BERT-based architecture consists of six Transformer blocks, each identical to the unidirectional Transformer block in the hollow Transformer. However, in the τLDR and D3PM parameterization, no masking is applied, and for the masked model, we only mask the d-th dimension of the input $\boldsymbol{x}_t$. Each attention layer employs eight attention heads, while each hidden layer has a dimensionality of 256. We use a dropout rate of 0.1 and set the embedding dimension to 64. After obtaining the final output from the last Transformer block, we feed it into a two-layer ResNet block [31]. This block is identical to the decoder Transformer block in the hollow Transformer architecture, except that no attention layer is applied here. The resulting model comprises approximately 0.5 million parameters.

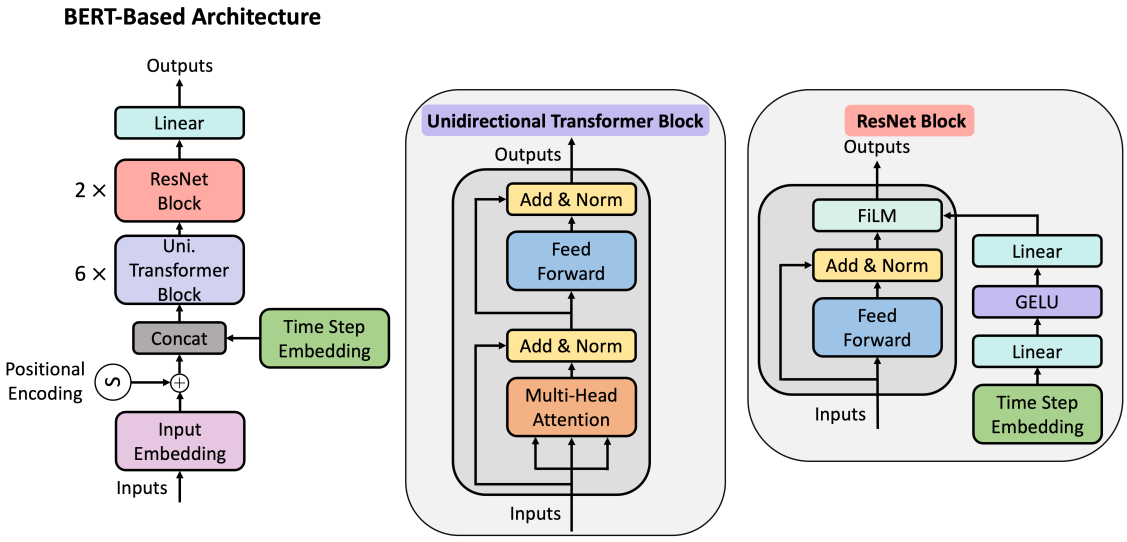


Figure 7.2.: Schematic illustration of the BERT-based architecture, based on the architecture used in [87].

For the hollow Transformer, we opt for two unidirectional Transformer blocks and one decoder Transformer block at the top of the architecture. Each block has eight attention heads, a hidden layer dimension of 256, and a dropout rate of 0.1. We set the embedding dimension to 64. Similar to our "BERT" model, the hollow Transformer has 0.5 million parameters.

Across all models, we maintain a constant learning rate of 1.5×10^{-4} and use the Adam optimizer [46] with standard settings. We clip gradient norms at a norm of three [69], and model training spans 200,000 iterations with a batch size of 128. For models trained with L_{CTEll}, L_{DTEll}, L_{CTEcrm}, and L_{CRMll}, we use a weight of $\lambda = 0.01$.

Following the procedure in [87], we employ a a uniform base rate matrix $R_b = \mathbb{1}\mathbb{1}^T - S\mathbb{I}$ for the forward process. Regarding $\beta(t)$, we use a cosine-style noise schedule [87], expressed as

$$\int_0^t \beta(v)\mathrm{d}v = -\sqrt{\cos\frac{\pi}{2}t} + 1.$$

The combination of a uniform base rate matrix and a cosine-style noise schedule leads to a uniform stationary distribution p_{ref} at time $t = T$ and ensures a reasonable noise level throughout the forward process to contribute to sample quality [65], as discussed earlier in Chapter 3. For our discrete-time model, we use uniform transition kernels, as shown in Equation 3.5. For $\beta(t)$, we also use a cosine noise schedule. Similarly, these choices lead to a uniform distribution at time $t = T$.

7.3.2. Evaluation Setup for the Spiral Dataset

In all experiments that focus on analyzing sample quality, i.e., the implicit vs. explicit parameterization experiment, the loss function evaluation, the cross-framework comparison, and the sampling study, we follow the approach in [87] and measure sample quality by comparing generated samples to actual data samples using the Maximum Mean Discrepancy (MMD) [29]. MMD is a kernel-based statistical metric for measuring the similarity between two distributions. It works by first mapping the data into a higher dimensional feature space $\mathcal{F}$ using a kernel function $\Phi(\boldsymbol{x}) \in \mathcal{F}$. We follow the evaluation in [87] and use an exponential Hamming kernel with a bandwidth of 0.1. It then extracts features from the data and computes their average difference. Given two distributions $\boldsymbol{x} \sim p(\boldsymbol{x})$ and $\boldsymbol{y} \sim q(\boldsymbol{y})$, the MMD is formally defined as

$$MMD(p(\boldsymbol{x}), q(\boldsymbol{y})) = \mathbb{E}_{p(\boldsymbol{x})}\left[k(\boldsymbol{x}, \boldsymbol{x})\right] - 2\mathbb{E}_{p(\boldsymbol{x}),q(\boldsymbol{y})}\left[k(\boldsymbol{x}, \boldsymbol{y})\right] + \mathbb{E}_{q(\boldsymbol{y})}\left[k(\boldsymbol{y}, \boldsymbol{y})\right],$$

where k represents the inner product of feature maps Φ. MMD values typically range from 0 to infinity, with smaller MMD values indicating greater similarity between distributions and larger values suggesting greater dissimilarity [29].

For the sample quality evaluation, we generate 4,000 samples and compare them to samples from the true data distribution using MMD. We repeat this process ten times, following the methodology outlined in [95]. In the implicit vs. explicit parameterization experiment, the loss function evaluation, and the cross-framework comparison, we fix the time step to $\tau = 0.002$

to maintain consistency across all simulation methods and frameworks. A τ value of 0.002 corresponds to approximately 500 NFE for the sampling methods used within continuous-time frameworks. Consequently, we train the D3PM model with $T = 500$ to ensure comparable NFEs for sampling in the discrete-time setting.

In our efficiency comparison between the frameworks and models, we use Euler sampling with a time step size of $\tau = 0.001$ to evaluate the sampling time and maximum generated samples in the continuous-time frameworks. A τ value of 0.001 equates to approximately 1000 neural network evaluations when using Euler sampling for the simulation of the reverse process. Hence, to ensure comparable NFEs during sampling, we train an additional D3PM model with $T = 1000$.

7.4. Setup Maze Dataset

For the categorical dataset, our work focuses on generating mazes, a challenge that has not been tackled to the best of our knowledge through generative modeling. These mazes are defined as 15×15 grids, where each cell in the grid initially represents either a wall or a path of the maze. We use the Growing Tree algorithm [82] to generate mazes with randomized entry and exit points on opposite sides for training. To better evaluate sample quality, we additionally incorporate a solution path into each maze using the Breadth-First Search algorithm [42]. As a result, each maze from this dataset has a dimensionality of $D = 15 \times 15 = 225$ and a state space size of $S = 3$. To eliminate any inherent ordinal structure, we scramble the ordering of the state space when mapping to $\mathbb{Z}$. Figure 7.3 illustrates a sample batch of these mazes, where walls are black, paths are white, and solution paths are grey.

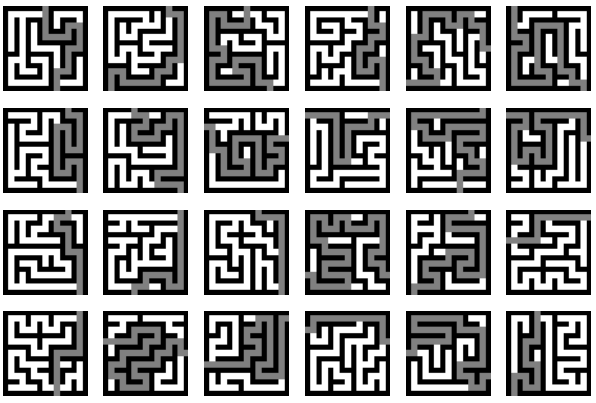


Figure 7.3.: Maze data samples.

7.4.1. Training Setup for the Maze Dataset

For the τLDR and D3PM model, we employ a customized 1D convolutional neural network (CNN), drawing inspiration from the network architecture taken for promoter generation in

[4]. This architecture, as depicted in Figure 7.4, consists of an input layer, 20 customized convolutional blocks, and a final output layer. In the input layer, the input data undergoes preprocessing. It is initially expanded into a higher-dimensional space through a learned embedding. Following the embedding, the data passes through a linear layer, followed by a Swish activation function [72], before being fed into the first convolutional block. Each convolutional block incorporates a time step embedding into its input, which is passed through a linear layer and subsequently added to the hidden state. The resulting data is normalized using group normalization [92] and passed through a 1D convolutional layer before being directed to an output layer. The output layer comprises a 1D convolutional layer, a GELU activation function, and another 1D convolutional layer. To ensure stability, the output is scaled to have a mean of zero. In total, the network has approximately 8 million parameters.

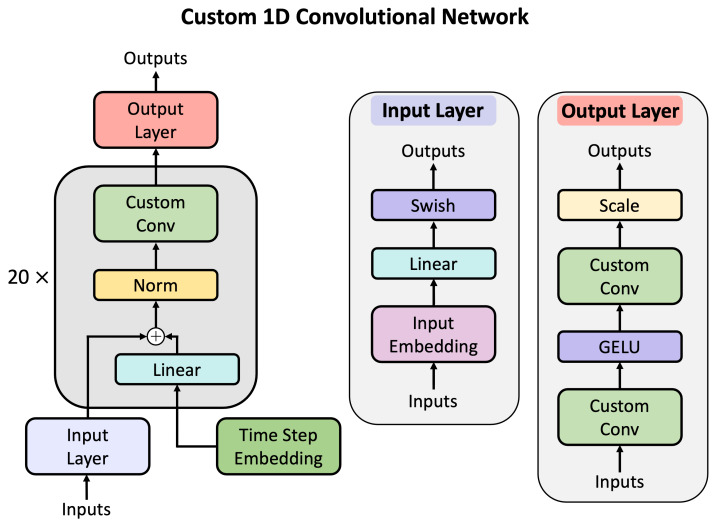


Figure 7.4.: Schematic illustration of the custom 1D convolutional architecture, used in the promoter generation task in [4].

In our efforts to parameterize the masked model, we experimented with various neural network architectures, including the previously described CNN and the same BERT-based architecture used for the spiral dataset. However, the computational demands of the masked model parameterization, requiring $O(D)$ evaluations of the neural network in each training step, make training reasonably sized neural networks infeasible due to our computational limitations. Consequently, we are left with the hollow Transformer as the only network option within the SDDM framework for this dataset. We configure the hollow Transformer with eight unidirectional Transformer blocks and two decoder blocks at the top. Each block has eight attention heads, hidden layers with 1024 dimensions, and a dropout rate of 0.1. We set the embedding dimension to 128. Similar to the custom CNN, the hollow Transformer has approximately 8 million parameters.

We train for 300,000 iterations with a batch size of 128 and a constant learning rate of 2×10^{-4} for all models. We clip gradients with a norm of three. Following the approaches described in [3, 7] for generating categorical data, we set $\lambda = 0.01$ for models trained using L_{CTEll} or L_{DTEll}. The same λ value of 0.01 is applied to models trained with L_{CTEcrm} or L_{CRMll}.

For the forward process, we employ the uniform base rate matrix and adopt the cosine-style noise schedule, which is consistent with the settings for our previous dataset. For the D3PM model, we also use the previously employed uniform transition kernels and a cosine noise schedule.

7.4.2. Evaluation Setup for the Maze Dataset

We use two metrics to assess the quality of our generated samples in the implicit vs. explicit parameterization experiment, the loss function evaluation, the cross-framework comparison, and the sampling study. First, we measure the proportion of generated mazes that are considered valid or correct, which we term as accuracy. Correctly generated mazes adhere to specific criteria:

- They have a single entry and exit on opposite sides of the maze.
- All other edges within the maze consist only of walls.
- Every valid maze must have a unique solution path, which is the shortest path from the entry to the exit.

To better illustrate this, Figure 7.5 provides visual examples of invalid mazes that deviate from these criteria. We refer to the mazes in this figure as incorrect or invalid for various reasons:

1. The first maze on the left lacks a border at the edge.
2. The second maze has multiple entries, one of which is not on the opposite side.
3. The third maze does not follow the direct path.
4. The last maze on the right contains an additional gray field not connected to the solution path.

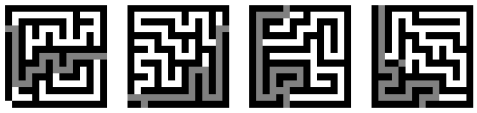


Figure 7.5.: Examples of invalid mazes.

Nevertheless, relying solely on accuracy may not provide a comprehensive evaluation of whether the distribution of our generated samples accurately approximates the true distribution. On one hand, even a single incorrect field in the maze can render it incorrect, as demonstrated in Figure 7.5. On the other hand, a model can achieve high accuracy by generating valid mazes. Nonetheless, these mazes may exhibit a distribution that significantly differs from the samples drawn from the original distribution. Figure 7.6 offers an example of this phenomenon. The mazes in this figure are all correct. However, compared to mazes from the true distribution,

as shown in Figure 7.3, they appear significantly simpler. The solution paths are much shorter and have fewer turns. Consequently, the distribution of these mazes differs significantly from the true distribution.

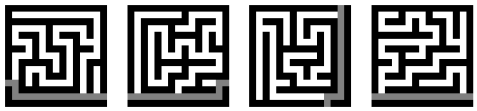


Figure 7.6.: Examples of valid mazes but differing distributions.

To address the concern of relying solely on accuracy as an evaluation metric, we use the Hellinger distance [55] as an additional measure for assessing the quality of the generated samples. Utilizing the Hellinger distance is inspired by the approach in [7], where it is used to evaluate the quality of generated categorical data samples. The Hellinger distance is designed to quantify the similarity between probability distributions. Mathematically, the Hellinger distance $HD(p(\boldsymbol{x}), q(\boldsymbol{x}))$ between two probability distributions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ is defined as

$$HD(p(\boldsymbol{x}), q(\boldsymbol{x})) = \frac{1}{\sqrt{2}} \left\| \sqrt{p(\boldsymbol{x})} - \sqrt{q(\boldsymbol{x})} \right\|_2 .$$

The values of $HD(p(\boldsymbol{x}), q(\boldsymbol{x}))$ range between 0 and 1, with 0 indicating perfect similarity and 1 maximum dissimilarity.

We conduct ten independent trials to evaluate sample quality and average the outcomes. In each trial, we evaluate the accuracy and Hellinger distance of 5,000 generated mazes. In the implicit vs. explicit parameterization experiment, the loss function evaluation, and the cross-framework comparison, we generate samples using a fixed $\tau = 0.001$ for all sampling methods. Consequently, we train the discrete-time model with $T = 1000$.

In our efficiency comparison, similar to the spiral dataset, we evaluate the sampling time and the maximum number of samples generated within the continuous-time frameworks using Euler sampling with a time step size of $\tau = 0.001$.

7.5. Setup MNIST Dataset

As our high-dimensional ordinal dataset, we use the MNIST dataset [56]. This dataset consists of 70,000 grayscale images of handwritten digits, divided into 60,000 images for training and 10,000 images for testing. Each image has a resolution of 28×28, translating into a dimensionality of $D = 28 \times 28 = 784$. In our modeling approach, we directly represent these images in the discrete data space. Each pixel within the image can take on values ranging from 0 to 255, effectively constituting our state space $S = 256$. Figure 7.7 visually represents a sample batch of data from this dataset.

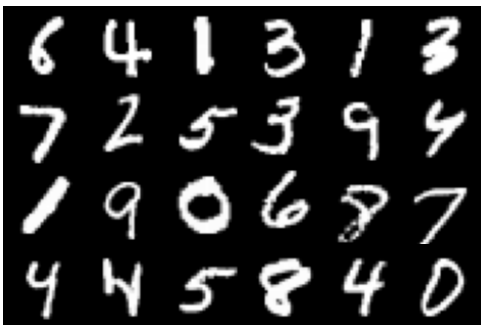


Figure 7.7.: MNIST data samples.

7.5.1. Training Setup for the MNIST Dataset

For the τLDR and D3PM model, we employ the standard U-Net architecture [76], following the design principles of the PixelCNN++ backbone [81].

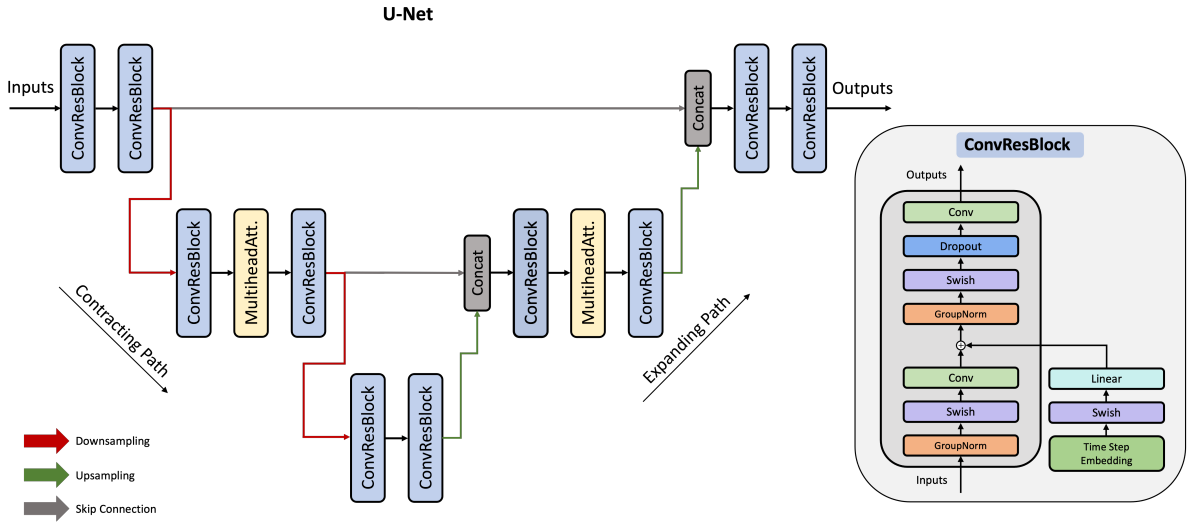


Figure 7.8.: Schematic illustration of the U-Net architecture.

As shown in Figure 7.8, the U-Net architecture consists of a contracting path and an expanding path. The contracting path contains encoder layers that capture contextual information and progressively reduce the spatial resolution of the input. Conversely, the expanding path contains decoder layers that gradually decode and upsample the encoded data to its original size and use the information from the contracting path via skip connections. In our adaption of this architecture, we utilize three feature map resolutions (28×28 to 7×7) in the contracting/-expanding paths. At each resolution, we use two convolutional residual blocks, each consisting of two convolutional layers, group normalization layers, dropout, and a Swish activation function. Additionally, we incorporate a multi-head self-attention block with eight attention heads between the residual blocks at the 14×14 resolution level [10]. The network receives time as input through sinusoidal position embedding, following the hollow Transformer's approach [90]. The time embedding is processed within each residual block through a Swish activation

followed by a linear layer before being added to the hidden state between the convolution operations [72]. In total, this network has about 14 million parameters.

Similar to the maze dataset, training masked models for the MNIST dataset is not feasible on our hardware, so we can only use the hollow Transformer in SDDM. Our hollow Transformer is configured with nine unidirectional Transformer blocks and two decoder blocks, each with eight attention heads, a hidden layer dimension of 512, an embedding dimension of 256, and a dropout rate of 0.1. The resulting hollow Transformer has approximately 14 million parameters.

Across all models, we maintain a constant learning rate of 2×10^{-4} using the Adam optimizer with standard settings. We clip gradient norms at a norm of one during training, which spans 600,000 iterations with a batch size of 64. We optimize the $L_{\text{CTEll}}/L_{\text{DTEll}}$ objective with $\lambda = 0.001$, following the approaches in [3, 7] for image modeling. For the models trained with L_{CTEcrm} and L_{CRMll}, we choose $\lambda = 0.01$.

For the forward noising process, we adopt the approach proposed in [7] in their image modeling task and use a discretized Gaussian transition matrix [3]. This choice encourages transitions between nearby states and facilitates the generation of realistic and coherent images by preserving smooth transitions and spatial relationships [7]. Appendix A.9 provides a systematic procedure for constructing this matrix. Similiar to [7], we incorporate $\beta(t) = 3 \cdot 100^t \log 100$ as our noise schedule. For the D3PM framework, we also utilize a discretized Gaussian transition matrix and employ a linear schedule as our noise schedule, following [3] in their image modeling task.

7.5.2. Evaluation Setup for the MNIST Dataset

To evaluate the quality of our generated samples in the implicit vs. explicit parameterization experiment, the loss function evaluation, the cross-framework comparison, and the sampling study, we employ the Fréchet Inception Distance (FID) [33], following [13, 94]. The FID score is a statistical metric used to quantify the similarity between the distributions of real and generated images by comparing their features extracted by an Inception-v3 neural network [88]. The comparison is done by fitting these features into two Gaussian distributions and measuring the Wasserstein-2 distance between them. Mathematically, the FID can be expressed as

$$\text{FID}(\boldsymbol{x}, \boldsymbol{z}) = \|\mu_{\boldsymbol{x}} - \mu_{\boldsymbol{z}}\|_2^2 + \text{Tr}\left(\Sigma_{\boldsymbol{x}} + \Sigma_{\boldsymbol{z}} - 2\left(\Sigma_{\boldsymbol{x}}\Sigma_{\boldsymbol{z}}\right)^{1/2}\right),$$

where Tr is the trace of a matrix, $\mu_{\boldsymbol{x}}, \mu_{\boldsymbol{z}}$ are mean and $\Sigma_{\boldsymbol{x}}, \Sigma_{\boldsymbol{z}}$ are covariance matrices fitted to the features of real images $\boldsymbol{x}$ and generated images $\boldsymbol{z}$ [20]. A lower FID score indicates that the generated images are more similar to real images in terms of their feature representations, suggesting a higher quality in the generated samples [33].

In addition to the FID score, we use the Inception Score (IS) [79] to measure our generated samples' diversity and class distinguishability. The IS has been widely used to assess the quality of generated images [79, 33, 3, 7]. It involves feeding the generated samples to a pre-trained Inception model, originally trained on the ImageNet dataset [16]. This classifier model evaluates the quality of generated images based on the diversity of categories they represent

and assigns higher scores to diverse and easily distinguishable images in terms of their content [79]. Mathematically, the IS is expressed as

$$\mathrm{IS}(p_{\mathrm{gen}}, p_{\mathrm{dis}}) = \exp\left(\mathbb{E}_{\boldsymbol{z}\sim p_{\mathrm{gen}}}\left[\mathcal{D}_{\mathrm{KL}}\left(p_{\mathrm{dis}}(\cdot|\boldsymbol{z}) \parallel \mathbb{E}_{\boldsymbol{z}\sim p_{\mathrm{gen}}}[p_{\mathrm{dis}}(\cdot|\boldsymbol{z})]\right)\right]\right),$$

where $\boldsymbol{z} \sim p_{\mathrm{gen}}$ are generated images and p_{dis} is the the classifier network. However, it is essential to note that the IS was primarily designed for evaluating generative models trained on complex and diverse datasets such as ImageNet [79]. When applying the IS to datasets like MNIST, which contain relatively simple and uniform data (handwritten digits), its interpretability and utility may be limited. To enhance the meaningfulness of the IS for MNIST, we follow the approach in [58] by feeding the generated samples to a classifier directly trained on the MNIST dataset instead of a model trained on ImageNet. This approach provides a more MNIST-specific context for interpreting IS scores, as the classifier is specifically tailored to the dataset. Nevertheless, the resulting IS depends on this specific classifier and is not comparable with IS scores from other classifiers.

We perform ten independent trials to evaluate sample quality and average the results. We compute the FID between 2,000 generated images and 2,000 real images from the training set in each trial. The IS is computed only on the 2,000 generated images. In the implicit vs. explicit parameterization experiment, the loss function evaluation, and the cross-framework comparison, we generate all images with a fixed time step of $\tau = 0.00067$.

For consistency in our efficiency comparison across the different frameworks, we employ Euler sampling with a time step of $\tau = 0.001$ within the continuous-time frameworks. Consequently, we train an additional D3PM model with $T = 1000$ to ensure comparability in the NFEs during sampling.

8. Experimental Results

This chapter presents the results of the experiments conducted based on the experimental setup described in the previous chapter. In particular, we analyze the parameterizations used in SDDM, evaluate the proposed loss functions within the SDDM and τLDR frameworks, compare the achieved sample quality and efficiency of these frameworks, and assess the sample quality of the presented sampling strategies at different time step values. Based on our findings, we provide recommendations to guide the selection of suitable modeling techniques for generating different types of discrete data.

8.1. Implicit vs. Explicit Parameterization

In our investigation, we begin by comparing the sample quality derived from implicitly parameterizing the conditional marginals $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d}) = \sum_{x_0^d} p_{0|t}^\theta(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|0}(X_t^d|x_0^d)$, with the sample quality resulting from explicitly predicting $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$ within SDDM. Throughout this section, we present the results in tables, where the sampling method that yields the best performance for each model is highlighted in bold. The overall best performance is presented in a boxed section. Generated samples can be found in Appendix B.

8.1.1. Results on the Spiral Dataset

Table 8.1 shows the comparison of spiral sample quality in terms of MMD, generated using masked models and hollow Transformers within SDDM, where each type includes one model configuration with explicit and another with implicit parameterization of $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$.

Table 8.1.: Comparison of MMD values (using an exponential Hamming kernel with a bandwidth of 0.1, in units of 1×10^{-4}) for spiral samples generated by the implicit and explicit masked models and hollow Transformers, each trained with L_{CRM}.

Metric: MMD (↓)		Sampler		
SDDM Model	Loss	Tau-Leaping	Euler	Analytical
Implicit Masked Model	L_{CRM}	0.137	**0.122**	0.133
Explicit Masked Model	L_{CRM}	0.225	**0.210**	—
Implicit Hollow Transformer	L_{CRM}	0.144	**0.136**	0.142
Explicit Hollow Transformer	L_{CRM}	0.163	**0.152**	—

For both the masked models and the hollow Transformers, the implicit parameterization provides superior sample quality in terms of MMD. Notably, while the implicit masked model slightly outperforms the implicit hollow Transformer, the explicit hollow Transformer surpasses the explicit masked model in performance. For all models, the samples generated by Euler sampling lead to the lowest MMD values.

8.1.2. Results on the Maze Dataset

In Tables 8.2 and 8.3, we evaluate the quality of the maze samples in terms of accuracy and Hellinger distance, generated by the implicit and explicit hollow Transformers within SDDM.

Table 8.2.: Comparison of accuracy scores for maze samples generated by the implicit and explicit hollow Transformers, each trained with L_{CRM}.

Metric: Accuracy (↑)		Sampler		
SDDM Model	Loss	Tau-Leaping	Euler	Analytical
Implicit Hollow Transformer	L_{CRM}	83%	**84**%	83%
Explicit Hollow Transformer	L_{CRM}	17%	**18**%	—

Table 8.3.: Comparison of Hellinger distances for maze samples generated by the implicit and explicit hollow Transformers, each trained with L_{CRM}.

Metric: Hellinger distance (↓)		Sampler		
SDDM Model	Loss	Tau-Leaping	Euler	Analytical
Implicit Hollow Transformer	L_{CRM}	0.0721	**0.0717**	0.0719
Explicit Hollow Transformer	L_{CRM}	0.08113	**0.0801**	—

Consistent with the previous results, the implicit parameterization of $p_t^\theta(X_t^d|\boldsymbol{x}^{\setminus d})$ yields better sample quality, achieving significantly higher accuracy and a lower Hellinger distance. Similarly, Euler sampling yields the best sample quality for both parameterization approaches.

8.1.3. Results on the MNIST Dataset

Tables 8.4 and 8.5 detail the quality of MNIST samples in terms of FID and IS, generated using the implicit and explicit hollow Transformers within SDDM.

Aligned with our previous findings, implicitly parameterizing the conditional marginals outperforms the explicit parameterization across both metrics. In particular, while using tau-leaping, Euler sampling, and midpoint tau-leaping leads to relatively poor FID values and Inception Scores for both parameterization approaches, using analytical sampling significantly improves the results within the implicit modeling paradigm. This observation highlights the advantage of the implicit parameterization in its flexibility to incorporate analytical sampling as an additional sampling scheme, demonstrating its ability to significantly improve sample quality.

Table 8.4.: Comparison of FID values for MNIST samples generated by the implicit and explicit hollow Transformers, each trained with L_{CRM}.

Metric: FID (↓)		Sampler			
SDDM Model	Loss	Tau-Leaping	Euler	Analytical	Midpoint TL
Implicit Hollow Transformer	L_{CRM}	66.74	94.29	**17.22**	64.36
Explicit Hollow Transformer	L_{CRM}	70.93	100.71	—	**67.82**

Table 8.5.: Comparison of IS for MNIST samples generated by the implicit and explicit hollow Transformers, each trained with L_{CRM}.

Metric: IS (↑)		Sampler			
SDDM Model	Loss	Tau-Leaping	Euler	Analytical	Midpoint TL
Implicit Hollow Transformer	L_{CRM}	6.23	5.65	**6.67**	6.36
Explicit Hollow Transformer	L_{CRM}	6.01	5.41	—	**6.24**

8.1.4. Summary and Recommendation

In summary, the implicit parameterization outperforms the explicit parameterization in terms of sample quality across all datasets. The performance advantage may stem from an effective utilization of the known expression of $q_{t|0}(X_t^d|x_0^d)$. While the explicit approach attempts to approximate the entire expression $q_t(X_t^d|\boldsymbol{x}_t^{\setminus d})$, the implicit method instead decomposes $q_t(X_t^d|\boldsymbol{x}_t^{\setminus d})$ into

$$q_t(X_t^d|\boldsymbol{x}_t^{\setminus d}) = \sum_{x_0^d} q_{0|t}(x_0^d|\boldsymbol{x}_t^{\setminus d}) q_{t|0}(X_t^d|x_0^d),$$

focusing on learning only $q_{0|t}(x_0^d|\boldsymbol{x}_t^{\setminus d})$ and integrating the known expression of $q_{t|0}(X_t^d|x_0^d)$ into our approximation $p_t^\theta(X_t^d|\boldsymbol{x}_t^{\setminus d})$. As indicated in [96], the approximation of an additional term, as required in the explicit parameterization, can introduce additional errors. Therefore, by circumventing approximating $q_{t|0}(X_t^d|x_0^d)$ and using the known expression of it, the implicit parameterization may lead to a more accurate approximation of q_t, which in turn could lead to better estimates of the reverse rates $\hat{R}_t$. Since we simulate the generative reverse process using these approximated reverse rates, a more accurate approximation of them leads to improved sample quality. Based on these results, we recommend parameterizing the conditional marginals implicitly, with the additional option of using analytical sampling for further potential performance improvement.

8.2. Loss Function Evaluation and Cross-Framework Comparison

In this section, we evaluate the proposed loss functions for the continuous-time frameworks and present the results of the cross-framework comparison in terms of sample quality. For clarity,

we only report the performance of the most effective sampler for each model configuration in our results. We highlight the model configuration with the best performing loss function within each framework in bold. The model configuration that performs best across all frameworks is highlighted in a box. Generated samples can be found in Appendix B.

8.2.1. Results on the Spiral Dataset

The quality of the generated spiral samples in terms of MMD can be seen in Table **??**. The reported results are obtained using Euler sampling, which consistently proves to be the best performing sampler across all models for this dataset.

Table 8.6.: Comparison of MMD values (using an exponential Hamming kernel with a bandwidth of 0.1, in units of 1×10^{-4}) for spiral samples generated by the masked model, hollow Transformer, τLDR model, and D3PM model, each trained using different loss functions.

Metric: MMD (↓)	Loss				
Model	L_{CRM}	L_{CTEcrm}	L_{CRMll}	$L_{\text{CTEll}}/L_{\text{DTEll}}$	L_{ll}
SDDM: Implicit Masked Model	**0.122**	0.473	0.221	—	—
SDDM: Implicit Hollow Transformer	**0.136**	0.463	0.189	—	—
τLDR: "BERT"	—	—	—	0.469	**0.210**
D3PM: "BERT"	—	—	—	**0.710**	—

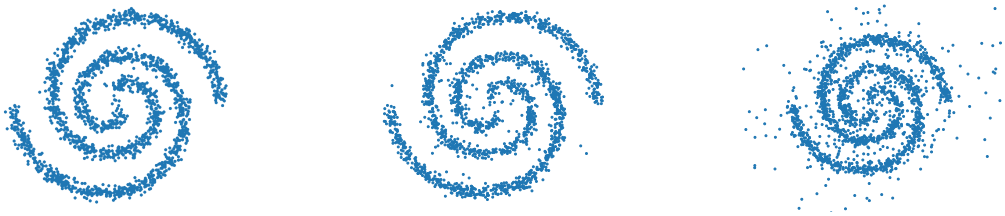
In our evaluation within the τLDR framework, we find that the model configuration trained using the L_{ll} loss significantly outperforms the one trained with L_{CTEll} in terms of MMD. This performance gap is further visually evident in Figure 8.1, where we show the generated samples from the τLDR models employing these respective loss functions. The samples generated by the L_{CTEll}-trained model, as shown in Figure 8.1a, display a significantly higher level of noise than the samples from the model trained with L_{ll}, illustrated in Figure 8.1b.

(a) Samples from the τLDR model trained with L_{CTEll}.

(b) Samples from the τLDR model trained with L_{ll}.

Figure 8.1.: Generated spiral samples with the τLDR model, using Euler sampling with a step size of $\tau = 0.002$. Each subfigure represents samples generated under different loss configurations: L_{CTEll}, L_{ll}, respectively.

Turning to the SDDM framework, our analysis reveals that the masked model and the hollow Transformer achieve their best sample quality in terms of MMD when trained with the L_{CRM} loss function. In particular, the L_{CRM}-trained masked model achieves the highest sample quality within SDDM, although the difference to the hollow Transformer trained with L_{CRM} is marginal. On the other hand, the L_{CTEcrm}-trained models perform the worst. This disparity in performance is similarly reflected in the visual quality of generated samples, as depicted in Figure 8.2. The samples from the hollow Transformer trained with L_{CTEcrm}, as shown in Figure 8.2c, are significantly noisier than those from the hollow Transformers trained with L_{CRM} or L_{CRMll}.



(a) Samples from implicit masked model trained with L_{CRM}.

(b) Samples from implicit masked model trained with L_{CRMll}.

(c) Samples from implicit masked model trained with L_{CTEcrm}.

Figure 8.2.: Generated spiral samples with the implicit masked model, using Euler sampling with a step size of $\tau = 0.002$. Each subfigure represents samples generated under different loss configurations: L_{CRM}, L_{CRMll}, and L_{CTEcrm}, respectively.

When comparing the performance across the frameworks, we observe that all model configurations from the continuous-time frameworks outperform the baseline discrete-time model. Among them, the implicit masked model using L_{CRM} for training emerges as the best performer.

8.2.2. Results on the Maze Dataset

In Tables **??** and 10.1, we analyze the quality of the generated maze samples in terms of accuracy and Hellinger distance. All models achieve their best performance when using Euler sampling for sample generation.

Table 8.7.: Comparison of accuracy scores for maze samples generated by the hollow Transformer, τLDR model, and D3PM model, each trained using different loss functions.

Metric: Accuracy (↑)	Loss				
Model	L_{CRM}	L_{CTEcrm}	L_{CRMll}	$L_{\mathrm{CTEll}}/L_{\mathrm{DTEll}}$	$L_{\mathrm{ll}^*}/L_{\mathrm{ll}}$
SDDM: Implicit Hollow Transformer	84%	42%	**85**%	—	85%
τLDR: CNN	—	—	—	52%	**94**%
D3PM: CNN	—	—	—	**87**%	—

Table 8.8.: Comparison of Hellinger distances for maze samples generated by the hollow Transformer, τLDR model and D3PM model, each trained using different loss functions.

Metric: Hellinger distance (↓)	Loss				
Model	L_{CRM}	L_{CTEcrm}	L_{CRMll}	$L_{\mathrm{CTEll}}/L_{\mathrm{DTEll}}$	$L_{\mathrm{ll}*}/L_{\mathrm{ll}}$
SDDM: Implicit Hollow Transformer	0.0717	0.0799	**0.0712**	—	0.714
τLDR: CNN	—	—	—	0.0783	**0.0693**
D3PM: CNN	—	—	—	**0.0709**	—

In our analysis of the τLDR framework, the model employing L_{ll} for training significantly outperforms the one using L_{CTEll} in terms of both accuracy and Hellinger distance. This superior performance is consistent with the results from the previous dataset.

Evaluating the results in SDDM, we find that the hollow Transformer trained with L_{CTEcrm} achieves the lowest sample quality, similar to the observations from the spiral dataset. However, in contrast to previous results, the hollow Transformer using L_{CRMll} for training performs better than the configuration trained with L_{CRM}, achieving a slightly higher accuracy and a lower Hellinger distance.

Motivated by the relative success of combining L_{CRM} with the auxiliary loss $L_{\mathrm{ll}*}$ in SDDM, along with the superiority of L_{ll} over L_{CTEll} in τLDR, we explored training a hollow Transformer model using only the auxiliary loss $L_{\mathrm{ll}*}$. From the results, detailed in Tables **??** and 10.1, we observe that the model trained with $L_{\mathrm{ll}*}$ exceeds the performance of the model using L_{CRM} and achieves comparable results to the L_{CRMll}-model. These findings indicate a potential effectiveness of the auxiliary loss for this dataset and the selected model configurations.

In our comparative analysis of framework performance, we find that the discrete-time model achieves better sample quality than all model configurations from the SDDM framework. Conversely, in the case of τLDR, although the discrete-time model shows better results than the τLDR model trained with L_{CTEll}, the model using L_{ll} excels over the discrete-time model, achieving the best overall sample quality. This superior performance of the L_{ll}-trained τLDR model can be attributed, on the one hand, to the effectiveness of the auxiliary loss, alongside the general advantages of a continuous-time framework. On the other hand, the notably lower performance of the hollow Transformers in SDDM, despite one hollow Transformer configuration utilizing a similar auxiliary loss, indicates that the neural network architecture also plays an important role.

Unlike SDDM, the τLDR framework allows for the flexible use of arbitrary neural network architectures. For this dataset, we adopted a customized CNN because of its ability to capture spatial relationships within the data [53]. This property could make CNNs particularly well suited for tasks such as maze generation, where the arrangement of walls, paths, and solution paths depends heavily on the spatial characteristics of neighboring cells. In addition, the translation-invariance property of CNNs allows them to recognize patterns regardless of their position within the input grid [57]. In maze generation, where identical maze patterns can manifest in different positions, the translation invariance could be an advantage for effective generalization.

In contrast, within the SDDM framework, we are limited to using the hollow Transformer due to the framework's architectural constraints and the computational demands of alternative architectures. While Transformers are powerful models for various tasks, they may not be the ideal choice for maze generation. Designed primarily for sequence-to-sequence tasks, Transformers may have difficulty capturing the spatial dependencies and grid-like structures inherent in maze data. Their self-attention mechanism, although powerful, does not inherently encode spatial information [90], and the employed positional encoding may not be sufficient to capture the underlying patterns in the data distribution. Therefore, the effective utilization of the custom CNN within the τLDR framework likely contributes to its superior performance compared to the models employed in SDDM.

8.2.3. Results on the MNIST Dataset

Tables 10.2 and 10.3 present the quality of the generated MNIST samples in terms of FID and IS. For the τLDR models, the use of midpoint tau-leaping, and for the hollow Transformers, the utilization of analytical sampling leads to the best performance.

Table 8.9.: Comparison of FID values for MNIST samples generated by the hollow Transformer, τLDR model, and D3PM model, each trained using different loss functions.

Metric: FID (↓)	Loss				
Model	L_{CRM}	L_{CTEcrm}	L_{CRMll}	$L_{\text{CTEll}}/L_{\text{DTEll}}$	$L_{\text{ll*}}/L_{\text{ll}}$
SDDM: Implicit Hollow Transformer	17.22	24.23	**13.93**	—	17.18
τLDR: U-Net	—	—	—	2.40	**1.75**
D3PM: U-Net	—	—	—	**1.88**	—

Table 8.10.: Comparison of IS for MNIST samples generated by the hollow Transformer, τLDR model, and D3PM model, each trained using different loss functions.

Metric: IS (↑)	Loss				
Model	L_{CRM}	L_{CTEcrm}	L_{CRMll}	$L_{\text{CTEll}}/L_{\text{DTEll}}$	$L_{\text{ll*}}/L_{\text{ll}}$
SDDM: Implicit Hollow Transformer	6.67	6.2	**7.32**	—	6.69
τLDR: U-Net	—	—	—	**8.8**	8.65
D3PM: U-Net	—	—	—	**8.6**	—

Analyzing the results in τLDR, we find mixed outcomes. On the one hand, the model trained with L_{ll} achieves a lower FID, indicating that the generated samples are closer to the true distribution. On the other hand, the model trained with L_{CTEll} achieves a higher Inception Score, suggesting greater diversity in the generated samples.

Shifting our focus to SDDM, we observe patterns consistent with previous studies. The L_{CTEcrm}-trained hollow Transformer demonstrates inferior sample quality to models trained with other loss functions. Moreover, mirroring the findings from the maze generation task, using L_{CRMll} for training leads to better results in terms of FID and IS than training with

L_{CRM}. Similarly, these results motivated us to train a hollow Transformer only with L_{ll^*}. However, as shown in Tables 10.2 and 10.3, in contrast to the maze dataset, training the hollow Transformer with L_{ll^*} does not lead to significant performance improvements, resulting in FID and IS values that are comparable to those obtained with L_{CRM}.

Comparing the performance across the frameworks, we analyze that the discrete-time model outperforms all models within SDDM, similar to the results for maze generation. In τLDR, while the model trained with L_{ll} outperforms the discrete-time model in terms of both FID and IS, the τLDR model trained with L_{CTEll} achieves a better IS but a worse FID value. Nonetheless, both the τLDR models and the D3PM model achieve better FID scores than the state-of-the-art models for the MNIST dataset, as summarized in Table 8.11.

Table 8.11.: FID Scores of the state-of-the-art models on the MNIST dataset [91].

Rank	Model	FID
1	τLDR + L_{ll} + Midpoint Tau-Leaping	1.75
2	D3PM + L_{DTEll}	1.88
3	τLDR + L_{CTEll}+ Midpoint Tau-Leaping	2.40
4	Sliced Iterative Normalizing Flows [13]	4.5
5	Generative Latent Flow + perceptual loss [94]	5.8
6	HypGan [54]	7.87

However, the significant performance contrast between SDDM and τLDR highlights the advantages of the τLDR framework in employing well-suited network architectures in high-dimensional data domains. U-Nets excel in image generation tasks due to their distinctive architecture, incorporating a contracting and an expanding path [76]. The contracting path focuses on feature extraction and global understanding, capturing high-level, global features that define the overall image structure [53]. Simultaneously, the expanding path maintains spatial details, preserving fine-grained features and pixel-wise relationships. This dual-path approach enables U-Nets to effectively capture local and global features, resulting in a good performance in image synthesis tasks [76, 3, 7].

Conversely, the SDDM framework forces us to use the hollow Transformer due to computational limitations. However, Transformers have inherent limitations when applied to high-dimensional image data such as the MNIST dataset. As mentioned previously, in the context of maze generation, Transformers often struggle to capture fine-grained spatial dependencies [90]. This limitation is even more pronounced for image datasets like MNIST, where preserving pixel-wise relationships is crucial for generating accurate images.

8.2.4. Summary and Recommendation

In conclusion, our findings within the τLDR framework reveal that, aside from the image modeling task where results are mixed, training with L_{ll} outperforms the use of L_{CTEll}. This observation indicates that despite training with L_{ll} leads to worse results in discrete time [3], it has the potential to enhance sample quality significantly in continuous-time modeling.

Therefore, we suggest considering training with L_{ll} in addition to the initially proposed loss L_{CTEll} in [7].

In SDDM, while integrating a similar auxiliary term L_{ll^*} into the categorical ratio matching loss L_{CRM} can improve sample quality, incorporating L_{CRM} into the CT-ELBO L_{eCTE} yields poor results across all datasets. Hence, we recommend using L_{CRM} and exploring the potential benefits of integrating L_{ll^*} in future work.

Furthermore, our results demonstrate that continuous-time models can achieve better sample quality than the discrete-time model across all datasets when equipped with suitable loss functions. While the SDDM framework outperforms τLDR for the low-dimensional binary spiral dataset by effectively leveraging the L_{CRM} loss function, its limitations become apparent for the higher-dimensional maze and MNIST datasets. In SDDM, the requirement for effective parameterization of the conditional marginals limits the choice of neural network architectures. Since masked models become prohibitively expensive to train with increasing data dimensionality, the hollow Transformer emerges as the only feasible network architecture for these higher-dimensional datasets within SDDM. However, despite its strengths in certain applications, the Transformer architecture may struggle to capture spatial dependencies and pixel-wise relationships. Conversely, the τLDR framework's flexibility in selecting arbitrary network architectures allows for the choice of models that are inherently more suited to the specific data application. Leveraging this adaptability in a continuous-time setting, the τLDR framework outperforms in higher dimensional data settings such as those presented by the maze and MNIST datasets. Therefore, we suggest exploring the use of the SDDM framework in lower-dimensional data applications but recommend using the τLDR framework for generating higher-dimensional data, particularly in datasets where the standard Transformer architecture encounters difficulties.

8.3. Efficiency Comparison

This section presents the analysis of the training and sampling efficiencies of both the continuous and discrete-time frameworks. Notably, the τLDR and D3PM frameworks utilize identical neural network architectures. In our case, the evaluation of the neural networks represents the primary bottleneck in training and simulation efficiency. As a result, the τLDR and D3PM models lead to similar outcomes, and we provide a consolidated summary of their results.

8.3.1. Results on the Spiral Dataset

In Table **??**, we analyze the training and sampling efficiency for the spiral dataset. While the hollow Transformer and the τLDR/D3PM model have comparable training and sampling times, the masked model demands significantly longer for both processes. This discrepancy arises from the masked model requiring $O(D)$ neural network evaluations in each training or sampling step. Since the sample quality of the implicit hollow Transformer trained with L_{CRM} is only slightly inferior to that of the best-performing model, the implicit masked model trained with L_{CRM},

but significantly more time efficient, this observation suggests that the hollow Transformer offers a more promising tradeoff between sample quality and computational efficiency for this dataset.

Table 8.12.: Comparison of training time (200,000 iterations with a batch size of 128), sampling time, and the maximum simultaneous sampling capacity for the masked model, the hollow Transformer, and the τLDR/D3PM model using Euler sampling ($\tau = 0.001$) on the spiral dataset.

Model	t_{Training}	Max Samples	t_{Sampling} (100,000 Samples)
SDDM: Hollow Transformer	2.5 h	130000	8 min
SDDM: Masked Model	20 h	100000	2.5 h
τLDR/D3PM: "BERT"	2 h	210000	6 min

8.3.2. Results on the Maze Dataset

Table **??** presents the training and sampling efficiency of the Hollow Transformer and the τLDR/D3PM models on the maze dataset.

Table 8.13.: Comparison of training time (300,000 iterations with a batch size of 128), sampling time, and the maximum simultaneous sampling capacity for the hollow Transformer and the τLDR/D3PM model using Euler sampling ($\tau = 0.001$) on the maze dataset.

Model	t_{Training}	Max Samples	t_{Sampling} (3900 Samples)
SDDM: Hollow Transformer	50 h	3900	22 min
τLDR/D3PM: CNN	20 h	28000	2 min

In contrast to the low-dimensional spiral dataset, where the hollow Transformer shows comparable efficiency to the τLDR/D3PM model, a significant difference is evident for this higher-dimensional maze dataset. Specifically, the hollow Transformer takes 2.5 times longer to train, necessitates eleven times longer for generating 3900 samples, and allows about seven times fewer samples to be generated simultaneously compared to the CNN in τLDR/D3PM. These inefficiencies of the hollow Transformer in training and sampling may be primarily attributed to the self-attention mechanism of the standard Transformer architecture. Self-attention has a time complexity of $O(D^2 d_{\text{model}})$ and memory complexity of $O(D^2 + D d_{\text{model}})$ [90]. Consequently, running two autoregressive unidirectional Transformers with multiple attention heads, along with an additional decoding Transformer block at the top of the architecture, can significantly increase the computational demands of the hollow Transformer as the dimensionality increases [44]. These differences in efficiency highlight the flexibility advantage of the τLDR/D3PM framework, allowing the use of more efficient neural network architectures tailored to the specific data application.

8.3.3. Results on the MNIST Dataset

Table **??** compares the training and sampling efficiency between the hollow Transformer and the τLDR/D3PM model on the MNIST dataset.

Table 8.14.: Comparison of training time (600,000 iterations with a batch size of 64), sampling time, and the maximum simultaneous sampling capacity for the hollow Transformer and the τLDR/D3PM model using Euler sampling ($\tau = 0.001$) on the MNIST dataset.

Model	t_{Training}	Max Samples	t_{Sampling} (200 Samples)
SDDM: Hollow Transformer	168 h	200	25 min
τLDR/D3PM: U-Net	20 h	2700	< 1 min

Since the MNIST dataset has a higher dimensionality of $D = 784$ compared to the maze dataset with a dimensionality of $D = 225$, we observe an even more pronounced difference in efficiency. The hollow Transformer takes about eight times longer to train, demands 25 times more time to generate 200 samples, and can generate about 13.5 times fewer samples simultaneously on our hardware compared to the U-Net model in τLDR/D3PM. This difference in efficiency underscores the limited scalability of SDDM and further emphasizes the flexibility advantage of the τLDR/D3PM framework.

8.3.4. Summary and Recommendation

In summary, for the low-dimensional spiral dataset, the hollow Transformer within SDDM offers a better balance between sample quality and efficiency than the best-performing model, the masked model. Consequently, for tasks with lower-dimensional data, where the standard Transformer architecture has no known limitations, we suggest first exploring the use of the hollow Transformer within SDDM to effectively balance these aspects. If the sample quality achieved by the hollow Transformer does not meet the requirements, it may be beneficial to consider the use of masked models as an alternative approach, albeit with potential efficiency drawbacks.

However, the SDDM framework's reliance on parameterizing conditional marginals becomes a significant limitation when dealing with higher dimensional data such as maze and MNIST. The conditional parameterization forces us to use the hollow Transformer architecture in these datasets, as alternative architectures for modeling this parameterization prove to be computationally infeasible with increasing data dimensionality. Nevertheless, Transformers such as the hollow Transformer also face computational challenges due to their architectural complexity, resulting in prolonged training and sampling times and substantial memory consumption in the case of the maze and MNIST datasets. In contrast, τLDR/D3PM's flexibility in selecting arbitrary neural network architectures provides a significant advantage in handling high-dimensional data by allowing us to employ more efficient neural network architectures tailored to the specific data application. This adaptability leads to superior efficiency compared to

SDDM, supporting our recommendation to use the τLDR framework for higher-dimensional data applications.

8.4. Sampling Study

This section evaluates the performance of the proposed sampling methods across various time steps, aiming to identify the best tradeoff between sample quality and simulation speed among the samplers for each dataset.

For evaluating the sampling methods, we employ the implicit hollow Transformer trained with L_{CRM} for the spiral and maze datasets. For the MNIST dataset, we use the U-Net trained with L_{CTEll}, which precludes us from assessing the performance of the analytical sampling scheme. However, due to computational constraints, generating a sufficient number of MNIST samples for evaluation with the hollow Transformer across different time step values is impractical.

8.4.1. Results on the Spiral Dataset

Figure 8.3 shows the comparison of sample quality in terms of MMD for generated spiral samples, using Euler sampling, tau-leaping, and analytical sampling across varying time steps. The x-axis (τ) and the y-axis (MMD) are plotted on a logarithmic scale.

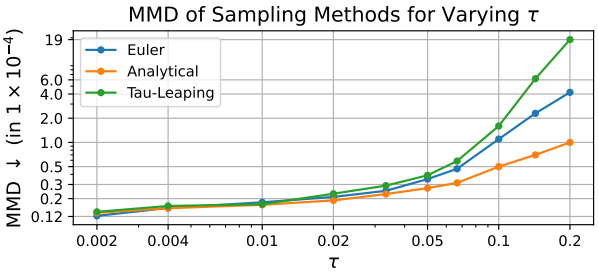


Figure 8.3.: Comparison of MMD values (using an exponential Hamming kernel with a bandwidth of 0.1, in units of 1×10^{-4}) for spiral samples generated by the implicit hollow Transformer, using Euler sampling, tau-leaping, and analytical sampling with varying time steps.

We observe that all sampling methods perform similarly for small time steps τ, though Euler sampling leads to the best sample quality measured in MMD for the smallest employed time step. However, when considering larger time steps ($\tau > 0.05$), analytical sampling consistently outperforms tau-leaping and Euler sampling. This observation suggests the potential robustness of analytical sampling, which aligns with the findings in [87]. The performance gap for larger time steps may be due to the fact that tau-leaping and Euler sampling require a small τ value to mitigate significant simulation errors [25, 87]. In contrast, analytical sampling directly approximates the analytical expression of the true reverse transition probabilities, completely avoiding any simulation errors [87].

In addition, Figure 8.3 highlights a potential difference in the quality-speed tradeoff between Euler sampling and tau-leaping when applied to this dataset. As τ approaches zero, both simulation methods converge to an exact simulation, leading to similar results. However, the inferior sample quality of tau-leaping for larger time steps may stem from its handling of multiple transitions within a single dimension in a single tau-leaping step for non-ordinal data. Specifically, as mentioned in Section 6.2.2, it was proposed in [7] to reject a state change for non-ordinal data within a dimension in a single tau-leaping step if that state change contains multiple transitions. Nevertheless, each of these rejected changes contributes to a deviation from an exact simulation since a state change would have occurred in an exact simulation. Such a deviation, in turn, negatively affects the sample quality. Now, since larger time steps τ lead to higher probabilities of multiple transitions occurring in one dimension in a single time step following a Poisson distribution $P_{ds} = \text{Poisson}\left(\tau \hat{R}_t^{\theta,d}(\boldsymbol{x}_t, s)\right)$, increasing the time step results in more state changes being rejected. This phenomenon is further illustrated in Figure 8.4, which shows the proportion of state changes that get rejected for containing multiple transitions. The proportion is averaged over the entire simulation of the reverse process for a given value of τ, underscoring how the rate of state rejections increases as τ grows. Consequently, the higher rejection rates at larger values of τ lead to a larger deviation of tau-leaping from an exact simulation, thus resulting in a worse performance. These findings suggest that tau-leaping may only perform well in non-ordinal data simulations when time step values are chosen to maintain reasonably low rejection rates. This potentially results in tau-leaping presenting a less favorable balance between simulation speed and sample quality compared to Euler and analytical sampling, as it needs to be simulated with smaller time steps to maintain a low rejection rate and achieve good performance.

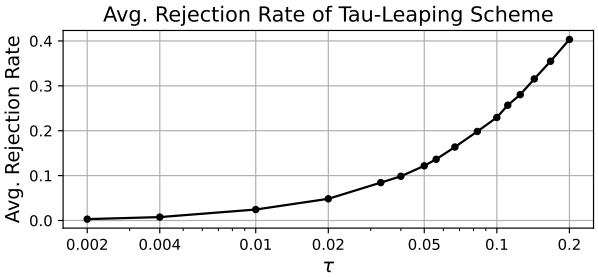


Figure 8.4.: Proportion of state changes that get rejected for containing multiple transitions within a single dimension for varying time step sizes. The proportion is averaged over the entire reverse process simulation, based on 3000 spiral samples.

To improve tau-leaping's performance at larger time steps, we propose not to reject state changes that contain multiple transitions in a single dimension for binary data. Unlike for general categorical data, in binary state spaces, multiple transitions in the same dimension inevitably lead to out-of-bounds transitions due to the limited presence of only two possible states. By allowing these multiple transitions while ensuring that the final state remains within bounds through clipping, we may approximate the exact simulation more closely than by outright rejecting these transitions.

Implementing the proposed adjustments, we observe notable improvements in sample quality in Figure 8.5, which extends the analysis of Figure 8.3 by including Tau-Leaping's performance

without the rejection of multiple transitions. Without rejections, tau-leaping achieves MMD values for $\tau > 0.05$ close to those obtained by analytical sampling. However, as the time step size decreases, the rejection rate diminishes, leading to comparable MMD values of tau-leaping with and without the rejection of multiple transitions.

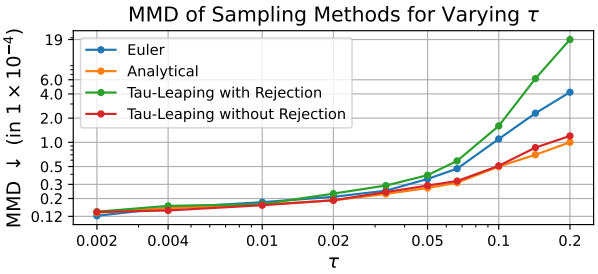
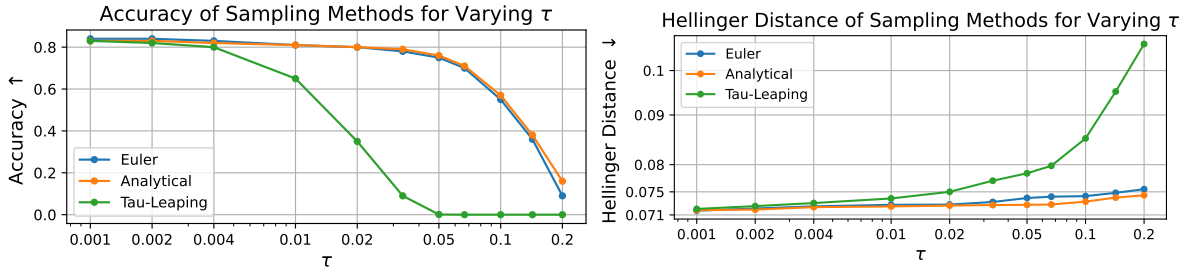


Figure 8.5.: Comparison of MMD values for spiral samples generated by the implicit hollow Transformer using Euler sampling, tau-leaping (with and without rejections), and analytical sampling, with varying time steps.

8.4.2. Results on the Maze Dataset

Figure 8.6a shows the accuracy and Figure 8.6b the Hellinger distance values for generated maze samples, using Euler sampling, tau-leaping, and analytical sampling with varying time steps. Both figures have a logarithmic x-axis, with Figure 8.6b also having a logarithmic y-axis.



(a) Accuracy

(b) Hellinger distance

Figure 8.6.: Comparison of accuracy and Hellinger distance for mazes samples generated by the implicit hollow Transformer, using Euler sampling, tau-leaping, and analytical sampling with varying time steps.

In line with the findings from the spiral dataset, we observe that all sampling methods exhibit similar sample quality in terms of accuracy and Hellinger distance for small time steps, with Euler sampling once again achieving the best performance at the smallest employed time step. Furthermore, analytical sampling also emerges as the most robust choice, demonstrating superior sample quality over tau-leaping and Euler sampling at larger time steps. However, while the performance gap between Euler sampling and analytical sampling is not as pronounced as observed in the binary spiral dataset, the disparity between Euler sampling and tau-leaping is considerably larger. The tau-leaping scheme only performs comparably to Euler and analytical sampling when $\tau \leq 0.004$. As depicted in Figure 8.7, which shows the average proportion of

changes that are rejected, it is evident that for $\tau > 0.004$, the rejection rate is reasonably high. Consequently, this leads to a greater deviation from an exact simulation of the reverse process for larger time steps. Thus, these findings could support the hypothesis that tau-leaping offers a less advantageous tradeoff between sample quality and simulation speed for non-ordinal data, attributed to the heightened rejection rate for larger time steps.

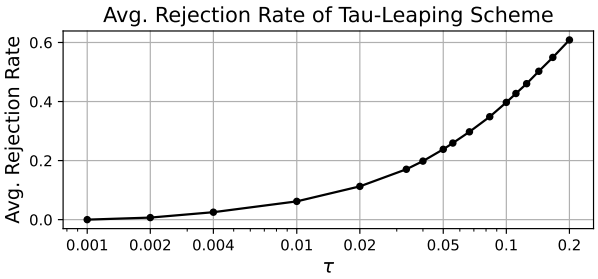


Figure 8.7.: Proportion of state changes that get rejected for containing multiple transitions within a single dimension for varying time step sizes. The proportion is averaged over the entire reverse process simulation, based on 3000 maze samples.

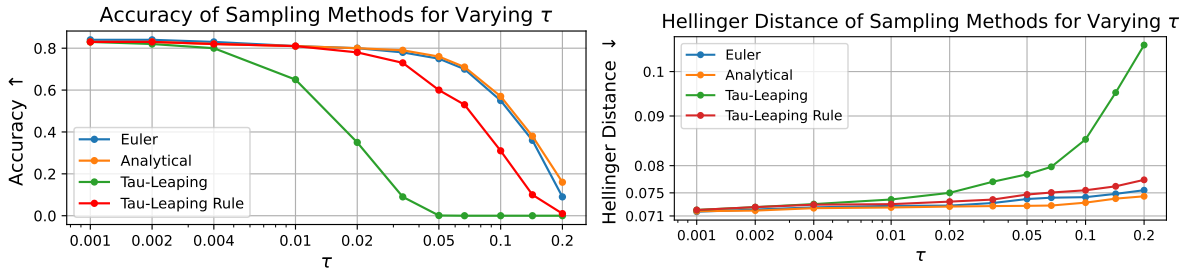
While allowing multiple transitions in one dimension in a single time step significantly improved tau-leaping's sample quality for larger time steps in the binary spiral dataset, in general categorical data, this approach renders the state-space mapping $\mathcal{X} \to \mathbb{Z}$ non-arbitrary. Instead, we propose alternative update rules to overcome this limitation and improve the quality-speed tradeoff of tau-leaping when dealing with categorical data. These rules, outlined below, aim to incorporate state changes that contain multiple transitions in a more meaningful way rather than outright rejecting them:

1. If multiple transitions occur in a single dimension d, but all transitions lead to a single state i ($P_{di} > 1$ and $P_{ds} = 0$ for all $s \in \mathcal{X}, s \neq i$), limit the number of transitions to state i to one (setting $P_{di} = 1$).

2. If multiple transitions occur in a single dimension d, leading to different states ($P_{ds} \geq 1$ holds for more than one $s \in \mathcal{X}$) and we have more transitions to one state i than to any other state in this dimension ($P_{di} > P_{ds}$ for all $s \in \mathcal{X}$, $s \neq i$), exclusively transition to state i (setting $P_{di} = 1$ and $P_{ds} = 0$ for all $s \in \mathcal{X}, s \neq i$).

3. If multiple transitions occur in a single dimension d, leading to different states ($P_{ds} \geq 1$ holds for more than one $s \in \mathcal{X}$), and no state has more transitions directed to it than any other, sample the new state based on the reverse rates and τ (similar to Euler sampling), then perform a transition to that state.

Implementing these update rules, we visualize the performance of tau-leaping alongside the other sampling procedures in Figures 8.8a and 8.8b. When these rules are applied, we observe a significant performance improvement in accuracy and Hellinger distance for $\tau > 0.004$. However, despite this improvement, the performance of this customized tau-leaping scheme still falls behind analytical and Euler sampling. Consequently, analytical and Euler sampling methods may provide a better balance between quality and simulation speed for categorical data.



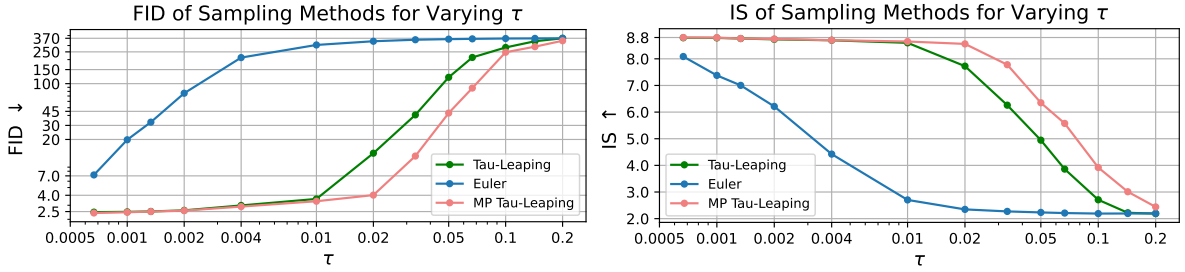
(a) Accuracy

(b) Hellinger distance

Figure 8.8.: Comparison of accuracy and Hellinger distance for mazes samples generated by the implicit hollow Transformer, using Euler sampling, tau-leaping, analytical sampling, and tau-leaping with a custom update rule, with varying time steps.

8.4.3. Results on the MNIST Dataset

Figure 8.9a shows the FID and Figure 8.9b the IS values for generated MNIST samples, using Euler sampling, tau-leaping, and midpoint tau-leaping with varying time steps. Both figures have logarithmic x-axes, and the y-axis is also logarithmic in Figure 8.9a.



(a) FID Values

(b) Inception Scores

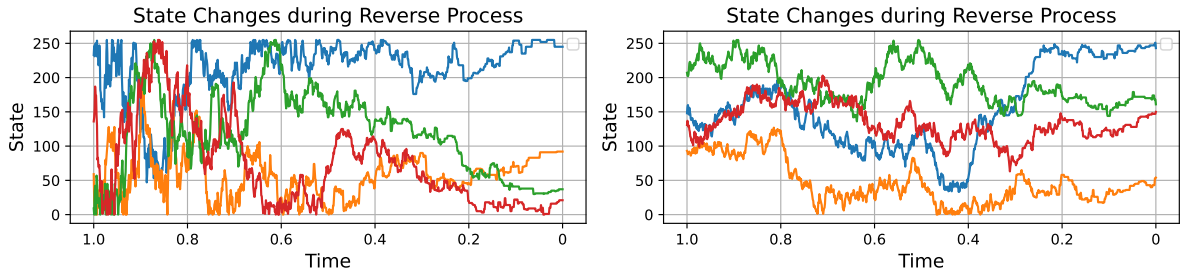
Figure 8.9.: Comparison of FID and IS for MNIST samples generated by the τLDR model, using Euler sampling, tau-leaping, and midpoint tau-leaping with varying time steps.

We analyze that midpoint tau-leaping consistently achieves superior sample quality in terms of FID and IS compared to tau-leaping and Euler sampling when $\tau > 0.01$. For $\tau \leq 0.01$, the FID and IS values of tau-leaping and midpoint tau-leaping become comparable. In contrast, Euler sampling shows minimal improvement in both metrics for $\tau > 0.01$. Even with the smallest employed step size of $\tau = 0.00067$, Euler sampling demonstrates significantly worse sample quality than tau-leaping and midpoint tau-leaping. In the following, we first examine the performance difference between Euler sampling and tau-leaping, since both simulation methods require the same NFEs. Afterward, we investigate the gap in sample quality at larger time steps between midpoint tau-leaping and tau-leaping.

Tau-Leaping vs. Euler Sampling: The observed performance gap between Euler sampling and tau-leaping indicates that Euler sampling is significantly less robust and requires smaller time steps to achieve comparable sample quality to tau-leaping for this ordinal dataset. The

discrepancy can be attributed to the unique characteristics of tau-leaping, permitting multiple transitions per dimension within a single step for ordinal data. In contrast, Euler sampling is limited to a single transition per dimension during a time step. Since both methods try to approximate an exact simulation, these characteristics could make Euler sampling a worse approximation for ordinal data for larger time steps and, thus, lead to a worse performance. This is because, in larger time intervals, the exact simulation likely involves multiple transitions in each dimension. Therefore, employing Euler sampling with a larger time step may provide a poor approximation of the exact simulation, as it only permits a single transition in each dimension per step. Conversely, Tau-Leaping's capability to facilitate multiple transitions in a single dimension for ordinal data may enable it to approximate the exact simulation more closely, leading to superior results at larger time steps.

From the observed state changes during the reverse process, we can also clearly see the differences between simulating the reverse process with tau-leaping and Euler sampling. Figures 8.10a and 8.10b illustrate the state progressions for selected representative dimensions during the reverse processes of tau-leaping and Euler sampling, simulated with $\tau = 0.00067$. We observe that during the initial phases of the reverse process, tau-leaping makes significantly broader state transitions compared to Euler sampling. These larger state transitions can be attributed to the higher proportion of state changes containing multiple transitions in tau-leaping during the early stages of the reverse process, as seen in Figure 8.11. This figure illustrates the proportion of state transitions that involve multiple transitions during a single tau-leaping step during the reverse process simulation, highlighting the distinct behavior of tau-leaping.



(a) Tau-Leaping (b) Euler sampling

Figure 8.10.: Visualization of state progressions in selected dimensions during the reverse process simulation using tau-leaping and Euler sampling with $\tau = 0.00067$.

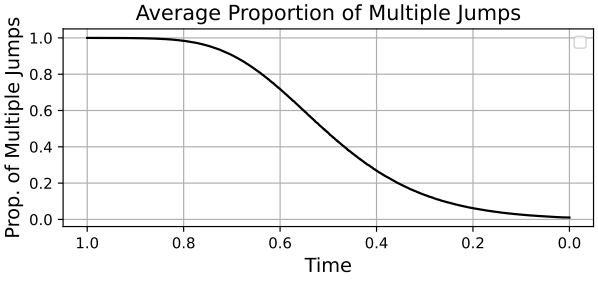
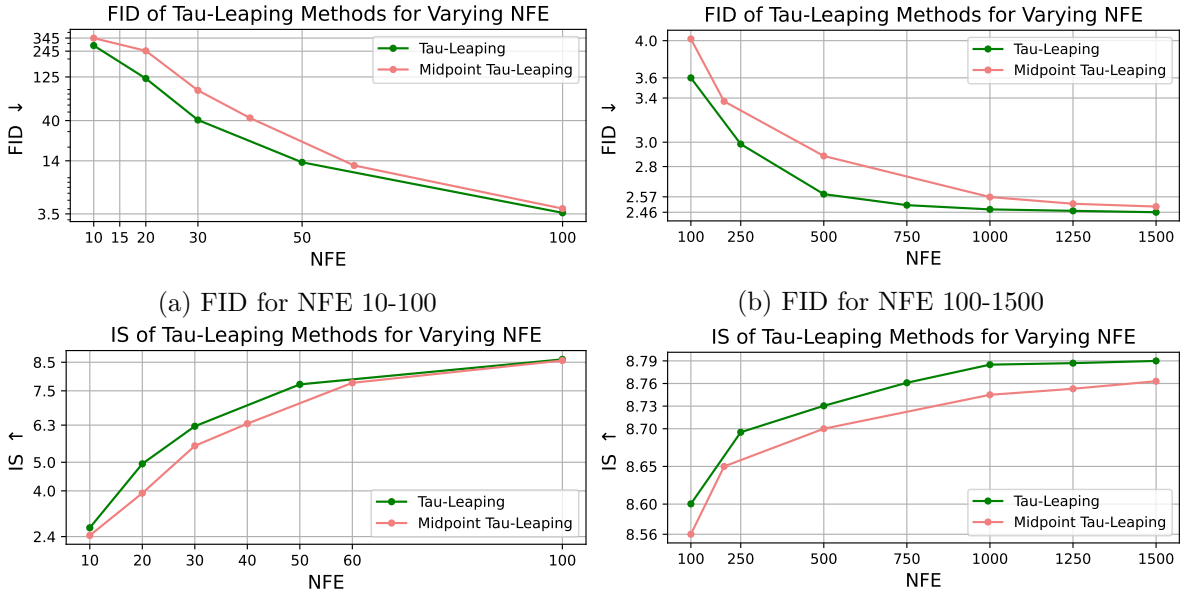


Figure 8.11.: Proportion of state changes that contain multiple transitions in a dimension during a single tau-leaping step during reverse process simulation with $\tau = 0.00067$. The proportion is averaged over a batch of 3000 MNIST samples.

Tau-Leaping vs. Midpoint Tau-Leaping: The superior performance of midpoint tau-leaping over tau-leaping for $\tau > 0.01$ may be due to its reduced simulation error compared to tau-leaping [2]. However, as τ decreases to or below 0.01, the performance gap between tau-leaping and midpoint tau-leaping diminishes, resulting in comparable sample quality. This observation can be attributed to the fact that, as τ approaches zero, both midpoint tau-leaping and tau-leaping converge to an exact simulation of the process, with midpoint tau-leaping no longer offering an accuracy advantage [73, 59]. Within our modeling configurations for this dataset, a τ value of 0.01 may already be sufficiently small to observe any significant performance difference between the two tau-leaping variants.

However, it is important to note that midpoint tau-leaping requires twice as many neural network evaluations as tau-leaping. Consequently, this leads to approximately twice the sampling time, as the neural network evaluation accounts for up to 99% of the required time for an iteration in the sampling process for our given modeling parameters. To further investigate whether midpoint tau-leaping offers an advantage in sample quality and speed, we plot the FID values and Inception Scores against the number of neural network evaluations in Figure 8.12. For better visualization, we have separated the results for the first 100 and the subsequent 1400 NFE for FID and IS. The y-axis in the FID plots in Figures 8.12b and 8.12c is in log space.



(a) FID for NFE 10-100

(b) FID for NFE 100-1500

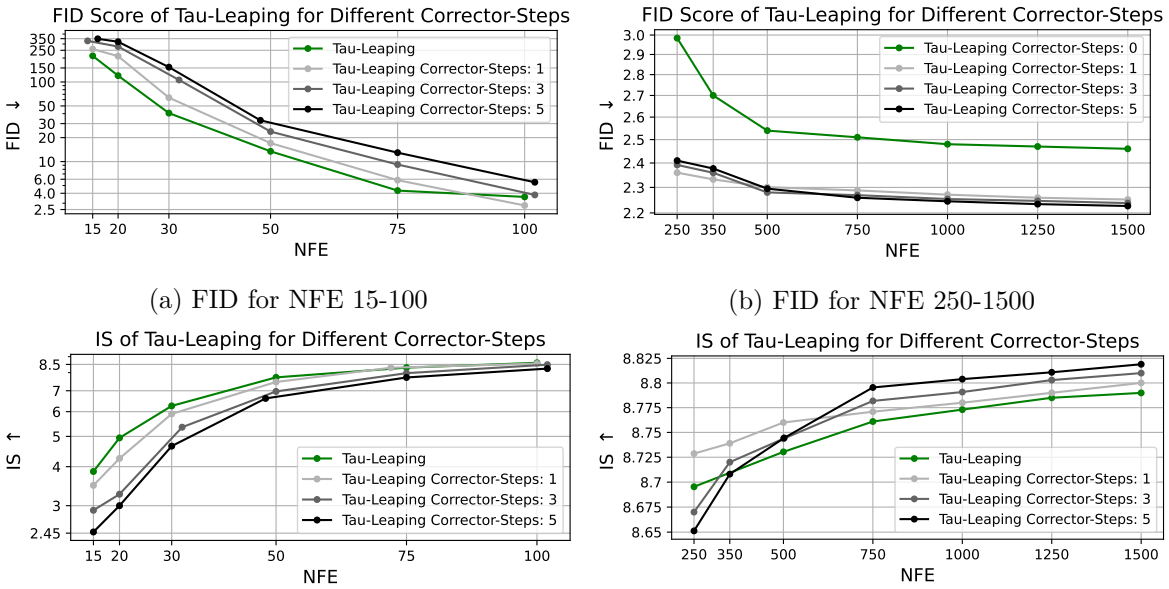
(c) IS for NFE 10-100

(d) IS for NFE 100-1500

Figure 8.12.: Comparison of FID and IS for MNIST samples generated by the τLDR model using tau-leaping and midpoint tau-leaping, plotted against a range of NFE.

We observe that tau-leaping consistently outperforms midpoint tau-leaping in terms of FID and IS when considering the same number of neural network evaluations. This observation suggests that tau-leaping may provide a more favorable balance between sample quality and simulation speed for this dataset and our chosen model configurations.

Predictor-Corrector Sampling: To explore alternative strategies for achieving a better balance between sample quality and simulation speed, we introduce different numbers of corrector steps within the tau-leaping scheme. Specifically, we introduce one, three, and five corrector steps after $t \leq 0.1T$ during the reverse process, following the approach recommended in [7]. Their findings suggested that additional corrector steps near the end of the reverse sampling process could significantly enhance sample quality with minimal computational cost. To evaluate each tau-leaping variant with different numbers of corrector steps, we follow the evaluation procedure outlined in Section 7.5.2. The results for up to 100 function evaluations are visualized in Figures 8.12a and 8.13c, while the results for 250 to 1500 NFE are shown in Figures 8.12b and 8.13d. In all figures, the y-axis is on a logarithmic scale.



(a) FID for NFE 15-100

(b) FID for NFE 250-1500

(c) IS for NFE 15-100

(d) IS for NFE 250-1500

Figure 8.13.: Comparison of FID and IS for MNIST samples generated by the τLDR model employing tau-leaping methods with additional corrector steps, plotted against a range of NFE.

We note that the optimal number of corrector steps depends on the number of function evaluations. For a smaller number of function evaluations, a smaller number of corrector steps appears to be more suitable. This phenomenon can be attributed to the increased value of τ required to maintain a fixed NFE when using a greater number of corrector steps [7]. Conversely, for a larger number of function evaluations corresponding to a smaller time step, employing a greater number of corrector steps tends to produce better outcomes. Overall, by integrating five corrector steps in each sample step after $t \leq 0.1T$, we are able to improve the FID score from 2.46 to 2.23 and the IS from 8.79 to 8.81 while keeping the NFE at 1500.

8.4.4. Summary and Recommendation

For the binary dataset, analytical sampling, Euler sampling, and tau-leaping — without rejecting multiple transitions in a single dimension — all yield favorable results. Tau-leaping and analytical sampling provide a good balance between sample quality and simulation speed, though Euler sampling leads to the best performance for the smallest employed time step. Consequently, we recommend experimenting with all three sampling methods for binary datasets, as testing each can reveal the most advantageous approach for specific dataset characteristics and performance goals.

For categorical data, we recommend using analytical sampling within the SDDM framework for its optimal tradeoff between sample quality and simulation speed. In addition, we advise using Euler sampling as it also achieves a good balance between these two aspects, alongside its superior performance at the smallest employed time step. Tau-leaping tends to underperform at larger time steps in categorical data due to the heightened rejection of transitions at larger time steps and the associated deviation from an exact simulation.

For our ordinal dataset, tau-leaping schemes offer a better speed-quality tradeoff than Euler sampling, likely due to their ability to facilitate multiple transitions within a single step. While midpoint tau-leaping demonstrates superior performance for larger time steps, it requires nearly double the simulation time for the same step size compared to standard tau-leaping. The main limitation in simulation speed, especially with large neural networks and high-dimensional data, is the required number of neural network evaluations. Consequently, for the MNIST dataset under our modeling parameters, midpoint tau-leaping does not improve the quality-speed balance. Therefore, we generally recommend using tau-leaping for ordinal data generation tasks. Yet, using midpoint tau-leaping could be beneficial in ordinal data scenarios where neural network evaluations are less costly and thus consume a smaller portion of the reverse process simulation time.

As a refinement to tau-leaping, adding extra corrector steps for simulations with finer time steps is advised. This adjustment can improve the quality-speed balance at smaller time steps by trading a smaller time step for a larger one, compensated by adding additional corrector steps. By employing this approach, we can maintain the same NFEs while achieving enhanced performance. Specifically, we suggest using fewer corrector steps at larger time steps, while more corrector steps can improve the balance between simulation speed and sample quality at smaller time steps.

9. Conclusion and Future Work

9.1. Conclusion

Given the limited research on continuous-time diffusion models in discrete state spaces, this thesis analyzed the current state-of-the-art continuous-time discrete diffusion frameworks, τLDR and SDDM. In our analysis, we conducted an experimental investigation of these frameworks to provide recommendations for selecting the most suitable framework and its modeling techniques for generating different types of discrete data. Through these recommendations, we aim to equip future researchers with a comprehensive basis and set of strategies for improving the generation of discrete data within continuous-time models. By doing so, we aspire to contribute to the development of more refined and precise techniques for modeling discrete data in future studies.

In our analysis, we extended beyond merely comparing the efficiency and performance in terms of sample quality of τLDR and SDDM as originally proposed. We broadened the investigation of each framework by exploring a wider range of modeling techniques. In particular, we compared explicit and implicit parameterizations of the conditional marginals within SDDM by evaluating their impact on sample quality. We assessed the performance of training models solely with an auxiliary loss function versus combining it with the CT-ELBO, as seen in τLDR. Furthermore, we adapted the CT-ELBO for training within the SDDM framework and trained models using a combination of the CT-ELBO and the categorical ratio matching loss to potentially improve sample quality. We also evaluated the effect of training SDDM models using a combination of the categorical ratio matching loss with a similar auxiliary loss on the quality of the generated samples. By making the sampling methods interchangeable between τLDR and SDDM, we expanded the range of available samplers beyond the original proposal of each framework. This expansion enabled a consistent within-framework comparison of sampler performance at different time steps, allowing us to explore the optimal balance between simulation speed and sample quality. In addition, we introduced midpoint tau-leaping as a new sampling method for continuous-time diffusion models and compared its performance with the other sampling methods in our ordinal data experiment.

Our results can be summarized as follows:
Implicit vs. Explicit Parameterization in SDDM: In our experiments within SDDM, the implicit parameterization of the conditional marginals consistently outperforms the explicit parameterization. Therefore, we recommend using the implicit modeling approach for its superior performance, with the option of using analytical sampling as an additional sampling method.

Loss Functions in τLDR: We found that solely training with the auxiliary term L_{ll} can enhance the sample quality when compared to training with a combination of L_{ll} and the CT-ELBO L_{eCTE} in the τLDR framework. Hence, it is beneficial to also consider training only with the auxiliary term in τLDR.

Loss Functions in SDDM: Combining the categorical ratio matching loss L_{CRM} with the auxiliary loss term L_{ll^*} can improve sample quality in categorical and ordinal data cases in SDDM. Conversely, combining L_{CRM} with the CT-ELBO L_{eCTE} does not lead to improvements. Therefore, we suggest using the categorical ratio matching loss for training and experimenting with auxiliary term integration to achieve optimal performance.

Cross-Framework and Efficiency Comparison: The choice between SDDM and τLDR depends on the data dimensionality and specific modeling requirements. The SDDM framework can offer good sample quality with appropriate training and simulation times in lower-dimensional data settings. However, the flexibility of τLDR to work with any neural network design makes it the preferred framework for addressing high-dimensional challenges. It allows the selection of well-suited architectures for specific data types, leading to superior performance and efficiency compared to SDDM, where the only viable choice in high-dimensional data is the hollow Transformer parameterization.

Sampling Study: For binary datasets, analytical sampling, Euler sampling, and tau-leaping (if multiple jumps are not rejected) all present viable options. For categorical data, we advise opting for analytical or Euler sampling due to their good tradeoffs between simulation speed and sample quality. In the case of ordinal data, tau-leaping variants proved more suitable due to their ability to jump multiple times in a single dimension, with tau-leaping offering the best balance between simulation speed and sample quality. As an alternative to "standard" tau-leaping, we recommend integrating additional corrector steps into tau-leaping for reverse process simulations with finer time steps. This adjustment can improve the quality-speed balance at smaller time steps by trading a smaller time step for a larger one, compensated by adding additional corrector steps.

By following these recommendations, researchers can effectively navigate the complexities of continuous-time diffusion modeling and tailor their approaches to specific datasets and objectives.

9.2. Future Work

Although the SDDM framework has shown promising results for low-dimensional data, a significant drawback lies in the architectural constraints that limit the framework's efficiency when dealing with high-dimensional data. To better accommodate high-dimensional data, one potential direction is to investigate the integration of more efficient attention mechanisms, such as masked axial attention [39, 37], within the hollow Transformer architecture. Axial attention reduces computational complexity by applying attention mechanisms independently along the axes of multidimensional data while preserving the model's expressiveness [37]. This

approach can enhance efficiency and ensures that the framework meets the requirements of masking the diagonals of the Jacobian of $p_t^\theta(\boldsymbol{X}_t|\boldsymbol{x}_t)$. In addition, to improve the performance of the SDDM framework on tasks involving image-like data, it may be advantageous to use techniques from Vision Transformers [19]. Vision Transformers divide the image into fixed-size patches, linearly embed each patch, add position embeddings, and then feed the resulting sequence of vectors into a standard Transformer encoder. Recent studies have shown that combining Vision Transformer architectures with continuous-time continuous state space diffusion models can outperform existing approaches for image generation [30]. Therefore, integrating Vision-Transformer-like architectures into SDDM may improve its performance on image modeling tasks. Furthermore, future researchers with more computational power could study the robustness of analytical sampling for generating image data such as MNIST.

In the context of τLDR, future researchers with access to greater computational resources could explore training models using only the auxiliary loss term L_{ll} and evaluate performance on standard benchmark datasets, such as CIFAR-10. Due to the high computational demands associated with processing such complex datasets, our current study was limited in this regard. In addition, to improve the quality-speed tradeoff, particularly for non-ordinal data, one could attempt to develop an analytical expression for reverse process sampling with the parameterization used in τLDR.

Furthermore, there are opportunities for future exploration in SDDM and τLDR regarding the sampling methods. One direction could be considering applying midpoint tau-leaping in ordinal data settings where network evaluation costs are lower and take up a smaller proportion of the sampling time. This approach may provide a better balance between sample quality and sampling time than tau-leaping in these settings. Another direction could be to explore integrating additional corrector steps within Euler sampling to further improve the performance in non-ordinal datasets without increasing the simulation time. Additionally, future researchers could investigate efficient step size selection methods [26, 8] for simulating the generative reverse process. These methods aim to reduce computational demands and improve efficiency by dynamically adjusting the granularity of time steps during the simulation. While we have attempted to implement step size selection methods, these methods required gradient computations of the reverse rates with respect to the input. This calculation proved to be computationally expensive and time-consuming in our scenarios, hindering their practical application. Exploring alternative approaches or optimizing existing methods for step size selection may lead to more efficient simulations of the reverse process in both SDDM and τLDR, ultimately enhancing the performance and scalability of these frameworks.

Bibliography

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

[2] David F. Anderson, Arnab Ganguly, and Thomas G. Kurtz. Error analysis of tau-leap simulation methods. *The Annals of Applied Probability*, 21(6):2226–2262, 2011.

[3] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[4] Pavel Avdeyev, Chenlai Shi, Yuhao Tan, Kseniia Dudnyk, and Jian Zhou. Dirichlet diffusion score model for biological sequence generation. In *International Conference on Machine Learning*, pages 1276–1301. PMLR, 2023.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[6] Daniel Brook. On the distinction between the conditional probability and the joint probability approaches in the specification of nearest-neighbour systems. *Biometrika*, 51(3):481–483, 1964.

[7] Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

[8] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics*, 124(4), 2006.

[9] Ricky TQ. Chen and David K. Duvenaud. Neural networks with cheap differential operators. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 864–872. PMLR, 2018.

[11] Hyungjin Chung, Byeongsu Sim, and Jong Chul Ye. Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12413–12422. IEEE, 2022.

[12] Thomas M. Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[13] Biwei Dai and Uros Seljak. Sliced iterative normalizing flows. In *International Conference on Machine Learning*, pages 2352–2364. PMLR, 2021.

[14] Hanjun Dai, Rishabh Singh, Bo Dai, Charles Sutton, and Dale Schuurmans. Learning discrete energy-based models via auxiliary-variable local exploration. *Advances in Neural Information Processing Systems*, 33:10443–10455, 2020.

[15] Ahmet Demirkaya, Jiasi Chen, and Samet Oymak. Exploring the role of loss functions in multiclass classification. In *2020 54th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5. IEEE, 2020.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[17] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[18] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped langevin diffusion. In *International Conference on Learning Representations*, 2021.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[20] D. C. Dowson and B. V. Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, 1982.

[21] William Feller. On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, page 403. University of California Press, 1949.

[22] Giulio Franzese, Simone Rossi, Lixuan Yang, Alessandro Finamore, Dario Rossi, Maurizio Filippone, and Pietro Michiardi. How much is enough? a study on diffusion times in score-based generative models. *Entropy*, 25(4):633, 2023.

[23] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889. PMLR, 2015.

[24] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *The Journal of Chemical Physics*, 2:403–434, 1976.

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

[26] Daniel T. Gillespie and Linda R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *The Journal of Chemical Physics*, 119(16):8229–8234, 2003.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[28] Frank Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.

[29] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[30] Ali Hatamizadeh, Jiaming Song, Guilin Liu, Jan Kautz, and Arash Vahdat. Diffit: Diffusion vision transformers for image generation. *arXiv preprint arXiv:2312.02139*, 2023.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[32] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[33] 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. *Advances in Neural Information Processing Systems*, 30, 2017.

[34] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[35] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[36] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[37] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[38] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances*

in Neural Information Processing Systems, 34:12454–12465, 2021.

[39] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 603–612. IEEE, 2019.

[40] Aapo Hyvärinen. Some extensions of score matching. *Computational Statistics & Data Analysis*, 51(5):2499–2512, 2007.

[41] Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4):695–708, 2005.

[42] Bonifacius Vicky Indriyono. Optimization of breadth-first search algorithm for path solutions in mazyin games. *International Journal of Artificial Intelligence & Robotics (IJAIR)*, 3(2):58–66, 2021.

[43] Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv e-prints*, page arXiv:2105, 2021.

[44] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International Conference on Algorithmic Learning Theory*, pages 597–619. PMLR, 2023.

[45] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[46] Diederik P. Kingma and Jimmy Ba. Adam. Adam: a method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[47] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in Neural Information Processing Systems*, 34:21696–21707, 2021.

[48] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.

[49] Peter E. Kloeden and Eckhard Platen. *Stochastic differential equations*. Springer, 1992.

[50] Andrei Kolmogoroff. Über die analytischen methoden in der wahrscheinlichkeitsrechnung. *Mathematische Annalen*, 104:415–458, 1931.

[51] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2020.

[52] Alex Krizhevsky and Geoffrey E. Hinton. Learning multiple layers of features from tiny

images. 2009.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[54] Diego Lazcano, Nicolás Fredes Franco, and Werner Creixell. Hgan: Hyperbolic generative adversarial network. *IEEE Access*, 9:96309–96320, 2021.

[55] Lucien Marie Le Cam and Grace Lo Yang. *Asymptotics in statistics: some basic concepts*. Springer Science & Business Media, 2000.

[56] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[57] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 991–999. IEEE, 2015.

[58] Chunyuan Li, Hao Liu, Changyou Chen, Yuchen Pu, Liqun Chen, Ricardo Henao, and Lawrence Carin. Alice: Towards understanding adversarial learning for joint distribution matching. *Advances in Neural Information Processing Systems*, 30, 2017.

[59] Tiejun Li. Analysis of explicit tau-leaping schemes for simulating chemically reacting systems. *Multiscale Modeling & Simulation*, 6(2):417–436, 2007.

[60] Phil Long and Rocco Servedio. Consistency versus realizable h-consistency for multiclass classification. In *International Conference on Machine Learning*, pages 801–809. PMLR, 2013.

[61] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion language modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

[62] Siwei Lyu. Interpretation and generalization of score matching. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 359–366, 2009.

[63] David MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[64] Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

[65] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, July 2021.

[66] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano

Ermon. Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics*, pages 4474–4484. PMLR, 2020.

[67] James R. Norris. *Markov Chains*. Number 2. Cambridge University Press, 1998.

[68] Bernt Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[69] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318. PMLR, 2013.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[71] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[72] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[73] Muruhan Rathinam, Linda R Petzold, Yang Cao, and Daniel T. Gillespie. Consistency and stability of tau-leaping schemes for chemical reaction systems. *Multiscale Modeling & Simulation*, 4(3):867–895, 2005.

[74] Andy Rindos, Steven Woolet, Ioannis Viniotis, and Kishor Trivedi. Exact methods for the transient analysis of nonhomogeneous continuous time markov chains. In *Computations with Markov Chains: Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains*, pages 121–133. Springer, 1995.

[75] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[76] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[77] Sheldon M. Ross. *Stochastic processes*. John Wiley & Sons, 1995.

[78] Sheldon M. Ross. *Introduction to probability models*. Academic Press, 2014.

[79] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in Neural Information Pro-*

cessing Systems, 29, 2016.

[80] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2021.

[81] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2016.

[82] Lars Schulze-Falck. Python-maze. `https://github.com/Turidus/Python-Maze/tree/master`, commit = d169357ca045a5da19f05c129e14879299f47db6, 2023.

[83] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2256–2265. PMLR, June 2015.

[84] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[85] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020.

[86] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

[87] Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. In *The Eleventh International Conference on Learning Representations*, 2023.

[88] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[89] Yee Whye Teh, Max Welling, Simon Osindero, and Geoffrey E. Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4(12):1235–1260, 2003.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[91] Papers with Code. State-of-the-art results on image generation on mnist. `https://paperswithcode.com/sota/image-generation-on-mnist`, year = 2024, note = Accessed: February 2024.

[92] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[93] Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. In *International Conference on Learning Representations*, 2021.

[94] Zhisheng Xiao, Qing Yan, and Yali Amit. Generative latent flow. *arXiv preprint arXiv:1905.10485*, 2019.

[95] Dinghuai Zhang, Nikolay Malkin, Zhen Liu, Alexandra Volokhova, Aaron Courville, and Yoshua Bengio. Generative flow networks for discrete probabilistic modeling. In *International Conference on Machine Learning*, pages 26412–26428. PMLR, 2022.

[96] Pengze Zhang, Hubery Yin, Chen Li, and Xiaohua Xie. Formulating discrete probability flow through optimal transport. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[97] Lingxiao Zhao, Xueying Ding, Lijun Yu, and Leman Akoglu. Improving and unifying discrete&continuous-time discrete denoising diffusion. *arXiv preprint arXiv:2402.03701*, 2024.

[98] Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Truncated diffusion probabilistic models and diffusion-based adversarial auto-encoders. In *The Eleventh International Conference on Learning Representations*, 2022.

A. Proofs

Unless explicitly stated otherwise, in the following proofs, we consider one-dimensional discrete data $\mathcal{X}$ with finite cardinality $S = |\mathcal{X}|$.

A.1. Proof of Time-Reversed CTMC

Proof of that the time reversal of a CTMC is also a CTMC, adapted from [87].

Let $u \in [0, T]$ be the time, and let us define the forward time process as $\{X_t\}_{t\in[0,T]}$ and the reverse time process as $\{\hat{X}_t\}_{t\in[0,T]} = \{X_{T-t}\}_{t\in[0,T]}$. Similarly, let $\mathcal{F}_u = \sigma\{X_v \in [0, u]\}$ be the σ-algebra of events of the forward time process from time 0 to time u and also denote $\hat{\mathcal{F}}_u = \sigma\{\hat{X}_v \in [0, u]\} = \sigma\{\hat{X}_v \in [u, T]\}$ as the σ-algebra of events of the reverse time process from time u to time T. Now, consider any $A \in \hat{\mathcal{F}}_{T-t}$, any $r, t \in [0, T]$ with $r < t$, and any $x_t, x_r \in \mathcal{X}$. Then

$$
\begin{aligned}
&P\left(\hat{X}_{T-r} = x_r | \hat{X}_{T-t} = x_t, A\right) \\
&= P\left(X_r = x_r | X_t = x_t, A\right) = \frac{P\left(X_r = x_r, X_t = x_t, A\right)}{P\left(X_t = x_t, A\right)} \\
&= \frac{P\left(A | X_r = x_r, X_t = x_t\right) P\left(X_t = x_t | X_r = x_r\right) P\left(X_r = x_r\right)}{P\left(A | X_t = x_t\right) p\left(X_t = x_t\right)} \\
&= P\left(X_t = x_t | X_r = x_r\right) \frac{P\left(X_r = x_r\right)}{P\left(X_t = x_t\right)}
\end{aligned}
$$

is independent of A by the Markov property of X. Substituting back into our probabilities of the diffusion process, we have

$$
P\left(X_t = x_t | X_r = x_r\right) \frac{P\left(X_r = x_r\right)}{P\left(X_t = x_t\right)} = q_{t|r}(x_t|x_r) \frac{q_r(x_r)}{q_t(x_t)} = q_{r|t}(x_r|x_t),
$$

which proves that the time reversal of a CTMC is also a CTMC.

A.2. Proof of Relationship between Rate Matrices

Proof of $\hat{R}_t(x, \tilde{x}) = R_t(\tilde{x}, x)\frac{q_t(\tilde{x})}{q_t(x)}$, adapted from [7].

Let us consider the time derivative of the time reversal process $\{\hat{X}_t\}_{t\in[0,T]}$. For x, $\tilde{x} \in \mathcal{X}$ with $x \neq \tilde{x}$, we get

$$\begin{aligned}\frac{\partial}{\partial t} q_{T-t|T-r}(\tilde{x}|x) &= \frac{\partial}{\partial t}\left(\frac{q_{T-t}(\tilde{x})}{q_{T-r}(x)} q_{T-r|T-t}(x|\tilde{x})\right) \\ &= \frac{\frac{\partial}{\partial t} q_{T-t}(\tilde{x})}{q_{T-r}(x)} q_{T-r|T-t}(x|\tilde{x}) + \frac{q_{T-t}(\tilde{x})}{q_{T-r}(x)} \frac{\partial}{\partial t} q_{T-r|T-t}(x|\tilde{x}).\end{aligned}$$

Since the time reversal $\hat{X}_t$ is also a CTMC (proof in Appendix A.1), we use the Kolmogorov forward equation for the first term:

$$\begin{aligned}\frac{\partial}{\partial t} q_{T-t}(\tilde{x}) &= \frac{\partial}{\partial t} \sum_{x_0 \in \mathcal{X}} \pi_{\text{data}}(x_0) q_{T-t|0}(\tilde{x}|x_0) \\ &= \sum_{x_0 \in \mathcal{X}} \pi_{\text{data}}(x_0) \frac{\partial}{\partial t} q_{T-t|0}(\tilde{x}|x_0) \\ &= \sum_{x_0 \in \mathcal{X}} \pi_{\text{data}}(x_0) \sum_y -q_{T-t|0}(y|x_0) R_{T-t}(y, \tilde{x}) \\ &= -\sum_y q_{T-t}(y) R_{T-t}(y, \tilde{x}).\end{aligned}$$

Since $\mathcal{X}$ is a finite space, the summation over y is finite. Letting $t \to r$ and using that $\lim_{t\to r} q_{T-r|T-t}(x|\tilde{x}) = 0$, we obtain

$$\lim_{t\to r} \frac{\frac{\partial}{\partial t} q_{T-t}(\tilde{x})}{q_{T-r}(x)} q_{T-r|T-t}(x|\tilde{x}) = \frac{-\sum_y q_{T-t}(y) R_{T-t}(y, \tilde{x})}{q_{T-r}(x)} q_{T-r|T-t}(x|\tilde{x}) = 0.$$

For the second term, we use the Kolmogorov backward equation and relabel it:

$$\begin{aligned}\frac{\partial}{\partial r} q_{t|r}(x|\tilde{x}) &= -\sum_{y\in\mathcal{X}} R_r(y, x) q_{t|r}(\tilde{x}|y) \\ \frac{\partial}{\partial T - t} q_{T-r|T-t}(x|\tilde{x}) &= -\sum_{y\in\mathcal{X}} R_{T-t}(y, x) q_{T-r|T-t}(\tilde{x}|y) \\ \frac{\partial}{\partial t} q_{T-r|T-t}(x|\tilde{x}) &= \sum_{y\in\mathcal{X}} R_{T-t}(y, x) q_{T-r|T-t}(\tilde{x}|y).\end{aligned}$$

Letting again $t \to r$, then we have

$$\lim_{t\to r} \frac{q_{T-t}(\tilde{x})}{q_{T-r}(x)} \frac{\partial}{\partial t} q_{T-r|T-t}(x|\tilde{x}) = \frac{q_{T-r}(\tilde{x})}{q_{T-r}(x) R_{T-r}(x, \tilde{x})}.$$

Using the property $\hat{R}_{T-r}(x, \tilde{x}) = \lim_{t\to r} q_{T-r|T-r}(\tilde{x}|x)$ and combining these two results, lead us to

$$\hat{R}_{T-r}(x, \tilde{x}) = \lim_{t\to r} q_{T-t|T-r}(\tilde{x}|x) = 0 + R_{T-r}(\tilde{x}, x) \frac{q_{T-r}(\tilde{x})}{q_{T-r}(x)}.$$

By relabeling the time $T - r$, we get

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

Further, we can write the ratio $\frac{q_t(\tilde{x})}{q_t(x)}$ as

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

which is equivalent to the used parameterization of the τLDR framework if we substitute it back in.

A.3. Proof of CT-ELBO Derivation

To derive the CT-ELBO, we provide an informal proof, adapted from [7], that only leverages fundamental concepts from CTMCs. We show how the discrete-time ELBO seamlessly transitions into the continuous-time ELBO by considering an infinite series of infinitesimally small timesteps in the discrete-time formulation.

We begin by considering a CTMC over the interval $\mathcal{T} = [0, T]$ and partitioning the time $\mathcal{T}$ as follows: $0 = t_0 < t_1 < \cdots < t_{k-1} < t_k < \cdots < t_{K-1} < t_K = T$, with each interval $h = t_k - t_{k-1}$ being of the same length. For clarity and simplicity in distinguishing from continuous time, we use the shorthand notation k for the discrete time steps t_k in the subscripts. This setup transforms our problem into a discrete-time Markov chain characterized by forward and reverse transition kernels, $q_{k+1|k}(x_{k+1}|x_k)$ and $p^\theta_{k|k+1}(x_k|x_{k+1})$, respectively. The negative ELBO in its discrete-time form is then expressed as

$$\begin{aligned}L_{\text{DTE}}(\theta) = \mathbb{E}_{\pi_{\text{data}}(x_0)}\Bigg[&\underbrace{\mathcal{D}_{\text{KL}}\left(q_{K|0}(x_K|x_0) \parallel p_{\text{ref}}(x_K)\right)}_{L_K} - \underbrace{\mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p^\theta_{0|1}(x_0|x_1)\right]}_{L_0} \\ &+ \sum_{k=1}^{K-1} \underbrace{\mathbb{E}_{q_{k+1|0}(x_{k+1}|x_0)}\left[\mathcal{D}_{\text{KL}}\left(q_{k|k+1,0}(x_k|x_{k+1}, x_0) \parallel p^\theta_{k|k+1}(x_t|x_{k+1})\right)\right]}_{L_k}\Bigg].\end{aligned}$$

In the following, we write the L_{DTE} in terms of the CTMC rate matrices and take the limit as $h \to 0$ and $K \to \infty$ to obtain the continuous-time ELBO.

Let us first consider one item from L_k

$$\begin{aligned}L_k &= \mathbb{E}_{q_{t+1|0}(x_{k+1}|x_0)}\left[\mathcal{D}_{\text{KL}}\left(q_{k|k+1,0}(x_k|x_{k+1}, x_0) \parallel p^\theta_{k|k+1}(x_k|x_{k+1})\right)\right] \\ &= -\mathbb{E}_{q_t(x_k)q_{k+1|k}(x_{k+1}|x_k)}\left[\log p^\theta_{k|k+1}(x_k|x_{k+1})\right] + C,\end{aligned}$$

where we have absorbed terms that do not depend on θ into C. Let us now write $\log p^\theta_{k|k+1}$ in terms of the reverse rates with the help of the definition of the infinitesimal transition

probability in the reverse process

$$\begin{aligned}\log p^{\theta}_{k|k+1}(x_k|x_{k+1}) &= \log\left(\delta_{x_k,x_{k+1}}\hat{R}^{\theta}_k(x_{k+1},x_k)h + o(h)\right)\\ &= \delta_{x_k,x_{k+1}}\log\left(1+\hat{R}^{\theta}_k(x_k,x_k)h + o(h)\right)\\ &+ (1-\delta_{x_k,x_{k+1}})\log\left(\hat{R}^{\theta}_k(x_{k+1},x_k)h + o(h)\right)\\ &= \delta_{x_k,x_{k+1}}\left(\hat{R}^{\theta}_k(x_k,x_k)h + o(h)\right)\\ &+ (1-\delta_{x_k,x_{k+1}})\log\left(\hat{R}^{\theta}_k(x_{k+1},x_k)h + o(h)\right),\end{aligned}$$

where we used the series expansion of $\log(1+z) = z - \frac{z^2}{z} + o(z^2)$ for $|z^2| \leq 1, z \neq 1$, to seperate the cases of $x_k = x_{k+1}$ and $x_k \neq x_{k+1}$. This expansion is valid in our case, as we can take h small enough for every finite $\hat{R}^{\theta}_k$ such that $|h\hat{R}^{\theta}_k| \leq 1$, $h\hat{R}^{\theta}_k \neq 1$ holds for every k. Subsequently, we substitute this expression of $\log p^{\theta}_{k|k+1}$ into L_k and explicitly expressing the expectation over $q_{k+1|k}(x_{k+1}|x_k) = \delta_{x_k,x_{k+1}} + R_k(x_k,x_{k+1})h + o(h)$ as sum

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

Further, we can extract $\hat{R}^{\theta}_k$ within the log term by reorganizing it as follows

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

where the first two terms are independent of θ and tend to 0 as $h \to 0$. After this rearrangement and by absorbing constant terms into C, we arrive at the following expression

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

Currently, the denoising model $p^{\theta}_{0|k}$ has to be evaluated for each term in the sum of $\sum_{x_{k+1}\neq x_k}$, which would require multiple forward passes on the neural network. To improve computational efficiency, we can construct a new distribution, $g_k(x_{k+1}|x_k)$, to sample from [7]. This distribution is defined as

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

where

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

normalizes the sum over all possible transitions. The distribution $g_k(x_{k+1}|x_k)$ gives the probability of transitioning from x_k to a specific x_{k+1}, given that we know a transition occurs at time step k. Implementing this distribution yields an updated expression for L_k:

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

This formulation streamlines the computational process by reducing the required neural network passes.

Examining the other terms in L_{DTE}, we have that L_K is not dependent on θ, since we do not learn the forward process and L_0 we expand here:

$$\begin{aligned}
\mathbb{E}_{q_{1|0}(x_1|x_0)}\left[\log p_{0|1}^\theta(x_0|x_1)\right] &= \{\delta_{x_1,x_0} + hR_1(x_0, x_1) + o(h)\}\log p_{0|1}^\theta(x_0|x_1) \\
&= \log p_{0|1}^\theta(x_0|x_0) + h\sum_{x_1} R_1(x_0, x_1)\log p_{0|1}^\theta(x_0|x_1) + o(h) \\
&= hR_1^\theta(x_0, x_0) + h\sum_{x_1} R_1(x_0, x_1)\log p_{0|1}^\theta(x_0|x_1) + o(h).
\end{aligned}$$

In summary, we have

$$\begin{aligned}
L_{\text{DTE}} = h\mathbb{E}_{\pi_{\text{data}}(x_0)q_{1|0}(x_1|x_0)}&\left[-R_1^\theta(x_0, x_0) + \sum_{x_1} R_1(x_0, x_1)\log p_{0|1}^\theta(x_0|x_1)\right] \\
&- h\sum_{k=1}^{K-1}\mathbb{E}_{q_k(x_k)g_k(x_{k+1}|x_k)}\left[\hat{R}_k^\theta(x_k, x_k) + \mathcal{Z}_k(x_k)\log\hat{R}_k^\theta(x_{k+1}, x_k)\right]. \\
&+ o(h) + C
\end{aligned}$$

If we now take the limit of L_{DTE} as $h \to 0$ and $K \to \infty$, we obtain the transition from L_{DTE} to L_{CTE}

$$\lim_{h\to 0} L_{\text{DTE}} = L_{\text{CTE}} = -\int_0^T \mathbb{E}_{q_t(x)g_t(y|x)}\left[\hat{R}_t^\theta(x, x) + \mathcal{Z}_t(x)\log\left(\hat{R}_t^\theta(y, x)\right)\right]\mathrm{d}t + C.$$

We can approximate the integral using the Monte Carlo sampling by treating it as an expectation over a uniform distribution spanning the time interval $(0, T)$. Additionally, we explicitly represent $\hat{R}_t^\theta(x, x)$ as the negative sum of off-diagonal elements in the row. This results in the following expression

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

A.4. Proof of One Forward Pass

Proof that $q_t(x)$ and $\sum_x q_t(x)g_t(y|x)$ are very similiar distributions, adapted from [7]:

$$\begin{aligned}
\sum_x q_t(x)g_t(y|x) &= \sum_x q_t(x)(1-\delta_{x,y})\frac{R_t(x,y)}{\sum_{x'\neq x} R_t(x,x')} \\
&\propto -q_t(y)R_t(y,x) + \sum_x q_t(x)R_t(x,y) \\
&= q_t(y)\sum_{x'\neq y} R_t(y,x') + \partial_t q_t(y) \\
&\propto q_t(y) + \frac{1}{\sum_{x'\neq y} R_t(y,x')}\partial_t q_t(y) \\
&= q_t(y) + \delta_t \partial_t q_t(y).
\end{aligned}$$

Utilizing the Kolmogorov forward equation for the transition in the third step:

$$\frac{\partial}{\partial t}q_t(x) = \sum_z q_t(z)R_t(z,x),$$

where δ_t is the time between two consecutive transitions, defined as $\frac{1}{\sum_{\hat{x}\neq y} R_t(y,\hat{x})}$. Consequently, the expression $\sum_x q_t(x)g_t(y|x)$ can be interpreted as $q_{t+\delta_t}(y)$, inferred through a first-order Taylor series approximation centered at $q_t(y)$.

A.5. Proof of Categorical Ratio Matching Simplification

Proof of the simplified form of the categorical ratio matching loss, based on [87]. Here, we consider D-dimensional data $\mathcal{X}^D$ with finite cardinality $S^D = |\mathcal{X}^D|$. The core concept behind this proof lies in the fact that the conditional distribution on the d-th dimension does not depend on the specific value x^d. In other words, the value of $q_t(X_t^d = s|\boldsymbol{x}_t^{\backslash d})\log p_t^\theta(X_t^d = s|\boldsymbol{x}_t^{\backslash d})$ remains unaffected by x^d. Consequently, we can express this as follows

$$\begin{aligned}
&\sum_{\boldsymbol{x}_t\in\mathcal{X}^D} q_t(\boldsymbol{x}_t)\sum_{d=1}^{D}\sum_{s\in\mathcal{X}} q_t(X_t^d = s|\boldsymbol{x}_t^{\backslash d})\log p_t^\theta(X_t^d = s|\boldsymbol{x}_t^{\backslash d}) \\
&= \sum_{d=1}^{D}\sum_{s\in\mathcal{X}}\sum_{\boldsymbol{x}_t\in\mathcal{X}^D} q_t(\boldsymbol{x}_t)\frac{q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s)}{\sum_{s'\in\mathcal{X}} q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s')}\log p_t^\theta(X_t^d = s|\boldsymbol{x}_t^{\backslash d}) \\
&= \sum_{d=1}^{D}\sum_{s\in\mathcal{X}}\sum_{\boldsymbol{x}_t^{\backslash d}}\sum_{s''\in\mathcal{X}} q_t(\boldsymbol{x}_t^{\backslash d}, s'')\frac{q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s)}{\sum_{s'\in\mathcal{X}} q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s')}\log p_t^\theta(X_t^d = s|\boldsymbol{x}_t^{\backslash d}) \\
&= \sum_{d=1}^{D}\sum_{s\in\mathcal{X}}\sum_{\boldsymbol{x}_t^{\backslash d}}\frac{q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s)}{\sum_{s'\in\mathcal{X}} q_t(\boldsymbol{x}_t^{\backslash d}, X_t^d = s')}\log p_t^\theta(X^d = s|\boldsymbol{x}_t^{\backslash d})\sum_{s''\in\mathcal{X}} q_t(\boldsymbol{x}_t^{\backslash d}, s'')
\end{aligned}$$

$$= \sum_{d=1}^{D} \sum_{s \in \mathcal{X}} \sum_{\boldsymbol{x}_t^{\setminus d}} q_t(\boldsymbol{x}_t^{\setminus d}, X_t^d = s) \log p_t^\theta(X_t^d = s | \boldsymbol{x}_t^{\setminus d})$$

$$= \sum_{\boldsymbol{x}_t \in \mathcal{X}^D} \sum_{d=1}^{D} q_t(\boldsymbol{x}_t) \log p_t^\theta(X_t^d = x_t^d | \boldsymbol{x}_t^{\setminus d}).$$

By substituting this result into the original score matching loss function, we obtain a simplified and tractable loss function:

$$L_{\text{CRM}}(\theta) = \int_0^T \sum_{\boldsymbol{x}_t \in \mathcal{X}^D} q_t(\boldsymbol{x}_t) \left[\sum_{d=1}^{D} - \log p_t^\theta(x_t^d | \boldsymbol{x}_t^{\setminus d}) \right] \mathrm{d}t.$$

A.6. Proof of Analytical Computation of Forward Transition Probabilities

This proof, adapted from [7], establishes that if R_t and $R_{t'}$ commute for all t, t', then the following holds:

$$q_{t|0}(j|i) = \left(\exp \left[\int_0^t R_v \mathrm{d}v \right] \right)_{ij}.$$

Suppose Q_t represents the transition probability matrix with $(Q_t)_{ij} = q_{t|0}(j|i)$. We can show that $Q_t = \exp\left(\int_0^t R_v \, \mathrm{d}v\right)$ serves as a solution to the Kolmogorov forward equation, which in matrix form is expressed as $\partial_t Q_t = Q_t R_t$. Representing the matrix exponential in sum form

$$Q_t = \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t R_v \mathrm{d}v \right)^k$$

$$= \mathbb{I} + \int_0^t R_v \mathrm{d}v + \frac{1}{2!} \left(\int_0^t R_v \mathrm{d}v \right)^2 + \frac{1}{3!} \left(\int_0^t R_v \mathrm{d}v \right)^3 + \dots$$

Next, differentiating and utilizing the fact that R_t, $R_{t'}$ commute

$$\partial_t Q_t = R_t + \int_0^t R_v \mathrm{d}v R_t + \frac{1}{2!} \left(\int_0^t R_v \mathrm{d}v \right)^2 R_t + \dots$$

$$= \left\{ \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t R_v \mathrm{d}v \right)^k \right\} R_t.$$

Achieving the commutative property involves selecting $R_t = \beta(t) R_b$, where $\beta(t)$ is a time-dependent scalar and R_b is a constant base matrix. Note, since we are considering here one-dimensional discrete data $\mathcal{X}$ with finite cardinality $S = |\mathcal{X}|$, it follows that both R_t and R_b are matrices in $\mathbb{R}^{S \times S}$.

We can leverage the eigendecomposition of $R_b = U\Lambda U^{-1}$ to efficiently compute Q_t:

$$\begin{aligned}
Q_t &= \exp\left(\int_0^t R_v \mathrm{d}v\right) \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} \left(\int_0^t R_b \beta(v) \mathrm{d}v\right)^k \\
&= \sum_{k=0}^{\infty} \frac{1}{k!} \left(U\Lambda U^{-1} \int_0^t \beta(v) \mathrm{d}v\right)^k \\
&= U \left\{\sum_{k=0}^{\infty} \frac{1}{k!} \left(\Lambda U^{-1} \int_0^t \beta(v) \mathrm{d}v\right)^k\right\} U^{-1} \\
&= U \exp\left[\Lambda \int_0^t \beta(v) \mathrm{d}v\right] U^{-1}.
\end{aligned}$$

As Λ is a diagonal matrix, the matrix exponential aligns with the element-wise exponential operation, rendering the final expression computationally tractable [7].

A.7. Proof of Factorization of Forward and Reverse Rates

Proof of the factorized form of the forward and reverse rates of a CTMC, based on [7]. Here, we consider D-dimensional data $\mathcal{X}^D$ with finite cardinality $S^D = |\mathcal{X}^D|$.

Let us assume that the transition probabilities $q_{t|r}(\boldsymbol{x}_t|\tilde{\boldsymbol{x}}_r)$ of a CTMC decompose into a product $\prod_{d=1}^{D} q_{t|r}(x_t^d|x_r^d)$, where $q_{t|r}(x_t^d|x_r^d)$ are the transition probabilities for individual one-dimensional CTMCs characterized by forward rates $R_t^d(\tilde{x}_t^d, x_t^d)$ [7]. We simplify our notation by omitting the time subscript t from $\boldsymbol{x}_t$ in the upcoming derivation. To establish a relationship between R_t and R_t^d, we employ the Kolmogorov forward equation:

$$\frac{\partial}{\partial t} q_{t|r}(\boldsymbol{x}|\tilde{\boldsymbol{x}}) = \sum_{\boldsymbol{y} \in \mathcal{X}^D} q_{t|r}(\boldsymbol{y}|\tilde{\boldsymbol{x}}) R_t(\boldsymbol{y}, \boldsymbol{x}),$$

and insert the factorized form of $q_{t|r}$ into the left-hand side

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

yielding the equation

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

This relationship holds for any factorizable transition $q_{t|r}$, including when $q_{t|r}(\boldsymbol{y}|\tilde{\boldsymbol{x}}) = \delta_{\boldsymbol{y},\boldsymbol{x}}$, leading us to the forward-rate equation

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

Incorporating this into the formula for the reverse rate, we derive for τLDR's parameterization

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

For SDDM's conditional parameterization, we have

$$\hat{R}_t(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \sum_{d=1}^{D} R_t^d(\tilde{x}^d, x^d) \delta_{\boldsymbol{x}^{\setminus d}, \tilde{\boldsymbol{x}}^{\setminus d}} \frac{p_t^{\theta}(\tilde{x}^d|\boldsymbol{x}^{\setminus d})}{p_t^{\theta}(x^d|\boldsymbol{x}^{\setminus d})}.$$

A.8. Proof of Stationary Distribution of Corrector Matrix

Proof that the corrector matrix $\hat{R}_t^c = R_t + \hat{R}_t$ has q_t as its stationary distribution. The proof is adapted from [7].

Applying the Kolmogorov forward equation to the forward process and the reverse process, where the latter has the same marginals as the forward but is time-reversed, yields the following equations:

For the forward process, we have

$$\frac{\partial}{\partial t} q_t(x_t) = \sum_{y \in \mathcal{X}} R_t(y, x_t) q_t(y).$$

The reverse process is given by

$$-\frac{\partial}{\partial t} q_t(x_t) = \sum_{y \in \mathcal{X}} \hat{R}_t(y, x_t) q_t(y).$$

Summing these two equations results in the following

$$\sum_{y \in \mathcal{X}} \left\{ R_t(y, x_t) + \hat{R}_t(y, x_t) \right\} q_t(y) = 0.$$

Hence, by comparison with the Kolmogorov equation, it can be concluded that $R_t + \hat{R}_t$ represents the rate matrix of a CTMC with an invariant distribution of q_t.

A.9. Construction of Discretized Gaussian Transition Matrix

The construction of this matrix involves a systematic procedure as follows:

To begin, we choose a desired stationary distribution, denoted as p_{ref}, and aim to construct a rate matrix R_b that encourages transitions to nearby states while maintaining p_{ref} as the stationary distribution. We start by selecting the discretized Gaussian distribution as our p_{ref}

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

where σ_0 represents the standard deviation. Next, we enforce a detailed balance between p_{ref} and the base rate matrix R_b to ensure that p_{ref} serves as the stationary distribution. This leads to the following condition on R_b:

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

This condition constrains the diagonal elements within R_b but does not fully specify the entire matrix. To complete the construction of the rate matrix, we assume that the mean (μ) is at the center of the state space, which is a viable choice for ordinal data [7]. We set off-diagonal terms to the right of the diagonal in the upper half of the rate matrix and off-diagonal terms to the left of the diagonal in the lower half to be equal to 1, following [7]. With this initial configuration, we iteratively define values for the remaining entries in R_b that the detailed balance condition has not determined. A visual representation of this process is provided below for an 8×8 rate matrix:

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

In the matrix, $\square$ represents values to be determined, $\triangle$ indicates values defined relative to other entries through the detailed balance condition, and $\cdot$ signifies diagonal entries equal to the negative off-diagonal row sum. To assign values to the $\square$ entries, we adopt the scheme proposed in [7]. In particular, each $\square$ in a row is set to $\exp\left(-\frac{i^2}{\sigma_l}\right)$, where i represents the distance from the '1' in that row, and σ_l serves as a hyperparameter defining the length scale in state space for typical transitions. This choice encourages transitions between nearby states at a length scale of σ_l, facilitating the generation of realistic and coherent images by preserving smooth transitions and spatial relationships [7]. In our image modeling task, we choose $\sigma_0 = 512$ and $\sigma_l = 6$.

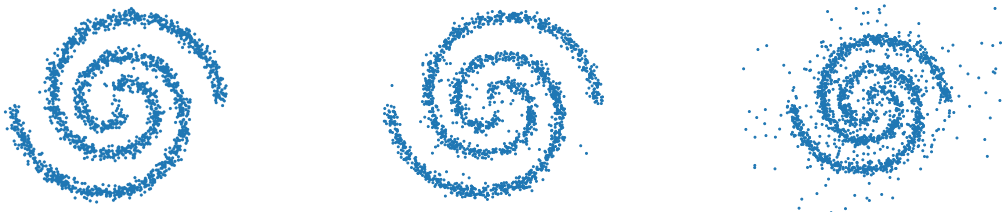
B. Generated Samples

B.1. Spiral Samples

(a) Samples from explicit hollow Transformer trained with L_{CRM}.

(b) Samples from explicit masked model trained with L_{CRM}.

Figure B.1.: Generated spiral samples by the explicit masked model and hollow Transformer, using Euler sampling with a step size of $\tau = 0.002$.

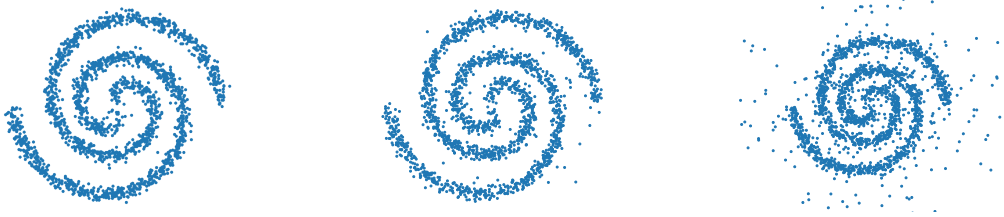


(a) Samples from implicit hollow Transformer trained with L_{CRM}.

(b) Samples from implicit hollow Transformer trained with L_{CRMll}.

(c) Samples from implicit hollow Transformer trained with L_{CTEcrm}.

Figure B.2.: Generated spiral samples by the implicit hollow Transformer, using Euler sampling with a step size of $\tau = 0.002$. Each subfigure represents samples generated under different loss configurations: L_{CRM}, L_{CRMll}, and L_{CTEcrm}, respectively.

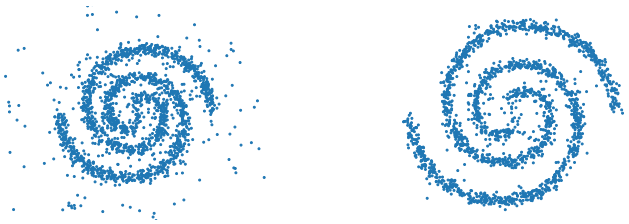


(a) Samples from implicit masked model trained with L_{CRM}.

(b) Samples from implicit masked model trained with L_{CRMll}.

(c) Samples from implicit masked model trained with L_{CTEcrm}.

Figure B.3.: Generated spiral samples by the implicit masked model, using Euler sampling with a step size of $\tau = 0.002$. Each subfigure represents samples generated under different loss configurations: L_{CRM}, L_{CRMll}, and L_{CTEcrm}, respectively.



(a) Samples from the τLDR model trained with L_{CTEll}.

(b) Samples from the τLDR model trained with L_{ll}.

Figure B.4.: Generated spiral samples by the τLDR model, using Euler sampling with a step size of $\tau = 0.002$. Each subfigure represents samples generated under different loss configurations: L_{CTEll}, L_{ll}, respectively.

Figure B.5.: Generated spiral samples using the D3PM model trained with L_{DTEll}.

B.2. Maze Samples

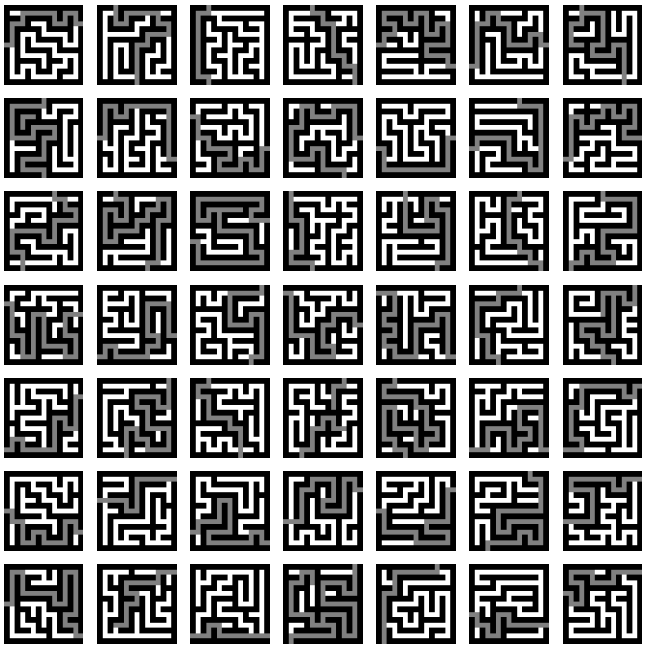


Figure B.6.: Generated maze samples by the implicit hollow Transformer trained with L_{CRM}, using Euler sampling with a step size of $\tau = 0.001$.

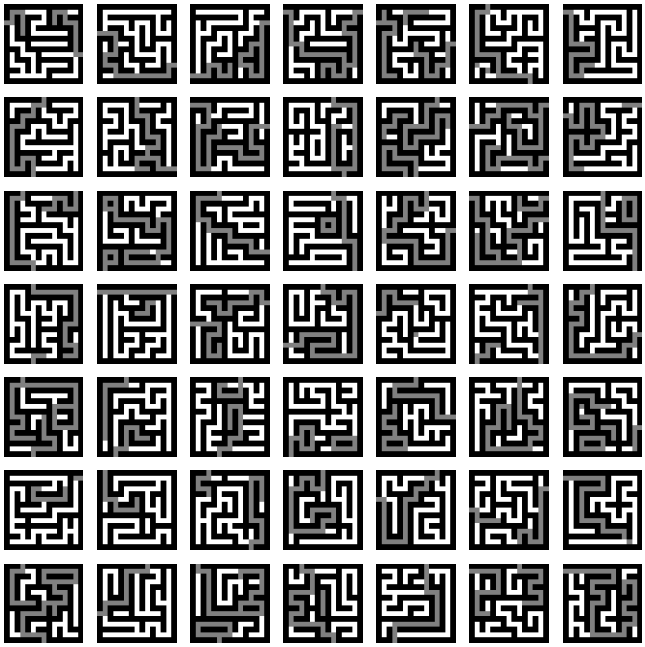


Figure B.7.: Generated maze samples by the explicit hollow Transformer trained with L_{CRM}, using Euler sampling with a step size of $\tau = 0.001$.

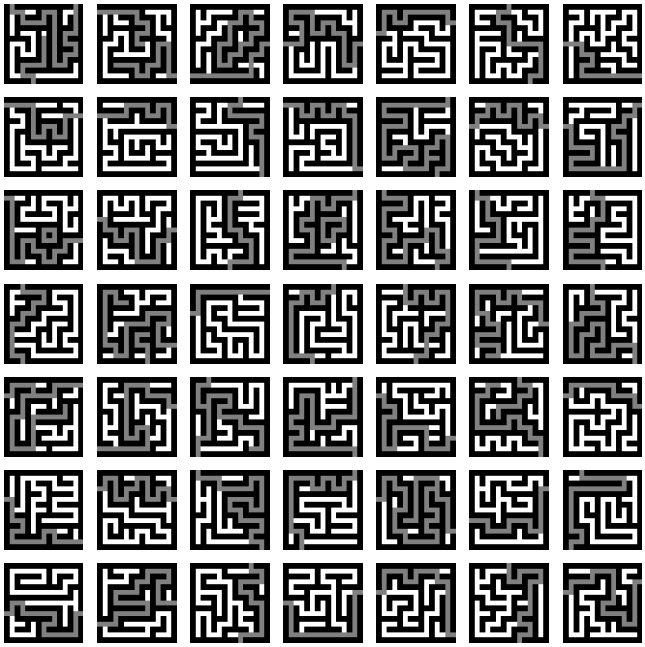


Figure B.8.: Generated maze samples by the implicit hollow Transformer trained with L_{CTEcrm}, using Euler sampling with a step size of $\tau = 0.001$.

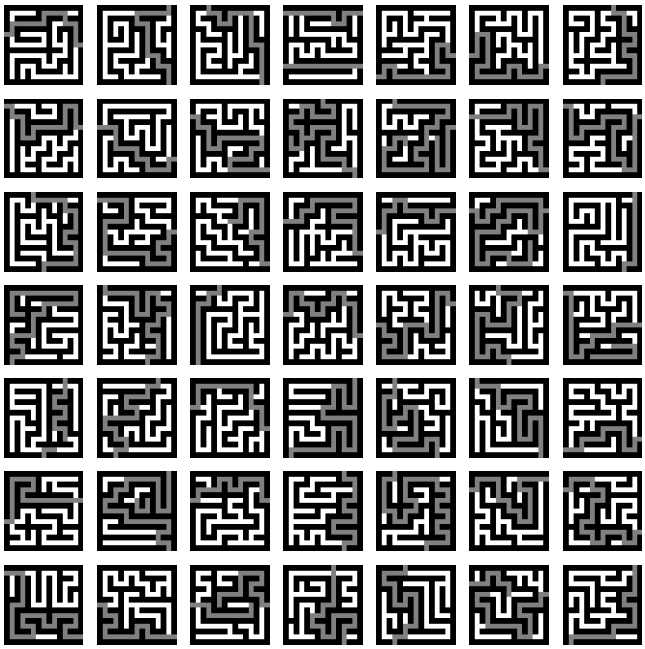


Figure B.9.: Generated maze samples by the implicit hollow Transformer trained with L_{CRMll}, using Euler sampling with a step size of $\tau = 0.001$.

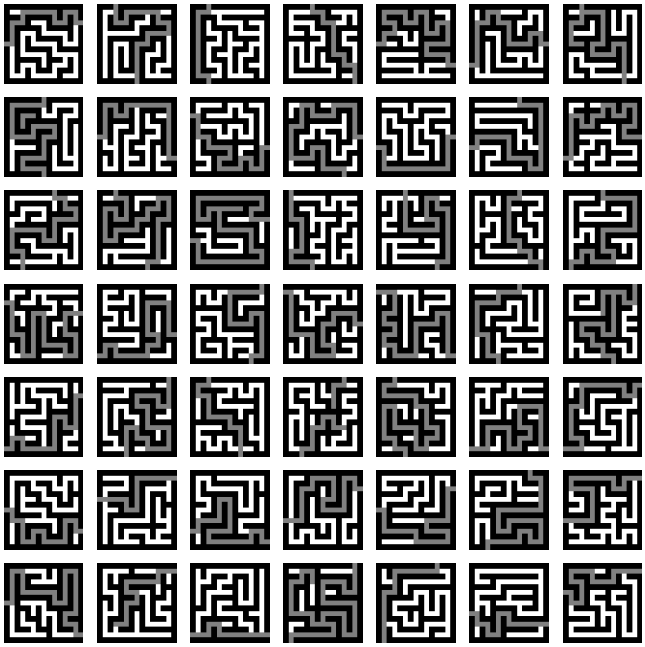


Figure B.10.: Generated maze samples by the τLDR model trained with L_{ll}, using Euler sampling with a step size of $\tau = 0.001$.

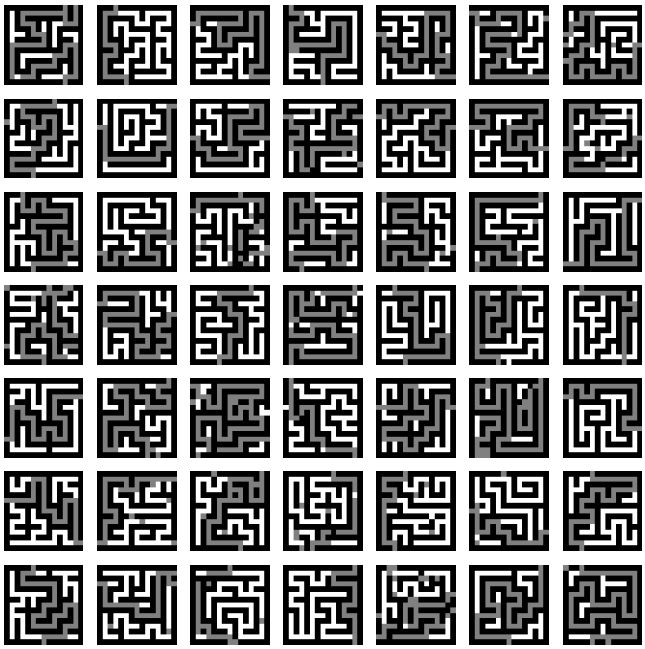


Figure B.11.: Generated maze samples by the τLDR model trained with L_{CTEll}, using Euler sampling with a step size of $\tau = 0.001$.

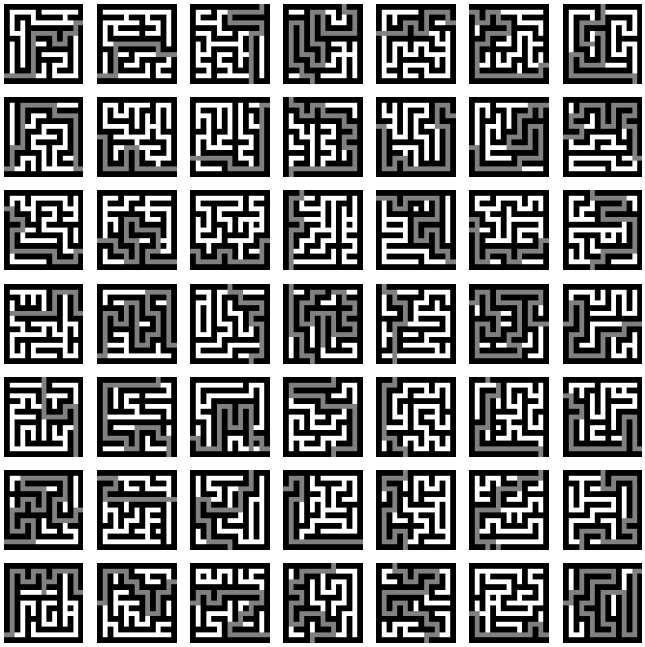


Figure B.12.: Generated maze samples by the D3PM model trained with L_{DTEll}.

B.3. MNIST Samples

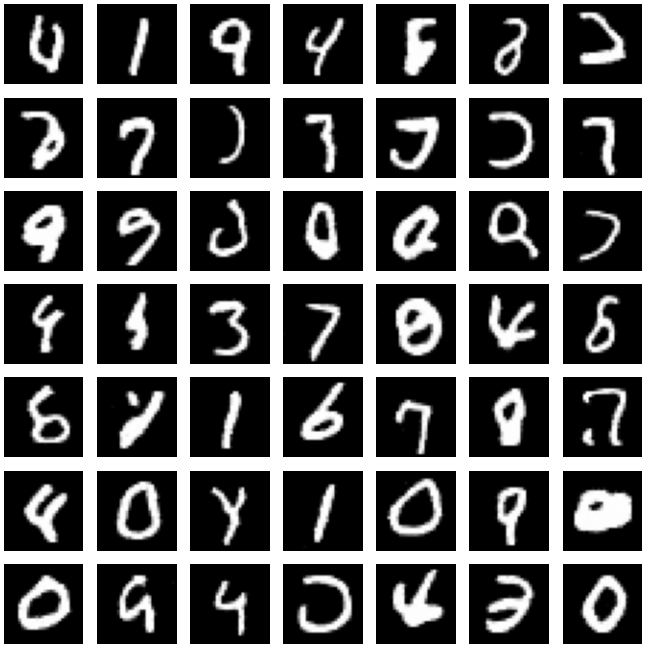


Figure B.13.: Generated MNIST samples by the implicit hollow Transformer trained with L_{CRM}, using analytical sampling with a step size of $\tau = 0.00067$.

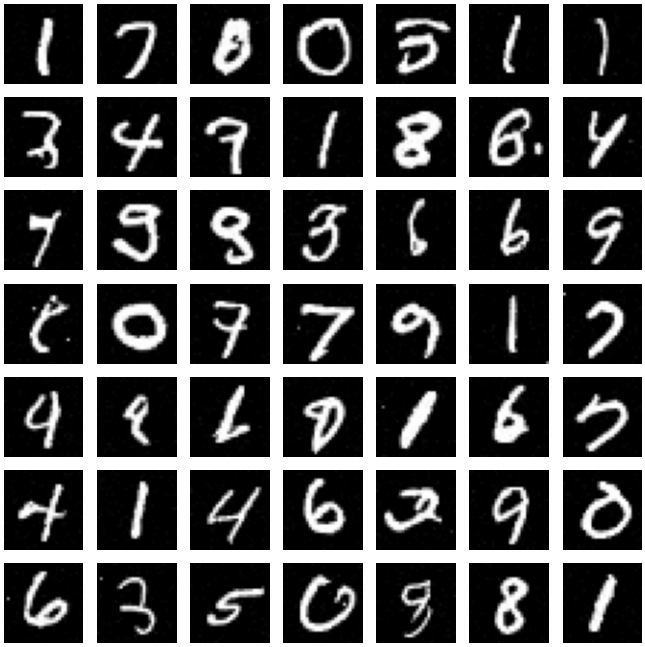


Figure B.14.: Generated MNIST samples by the explicit hollow Transformer trained with L_{CRM}, using midpoint tau-leaping with a step size of $\tau = 0.00067$.

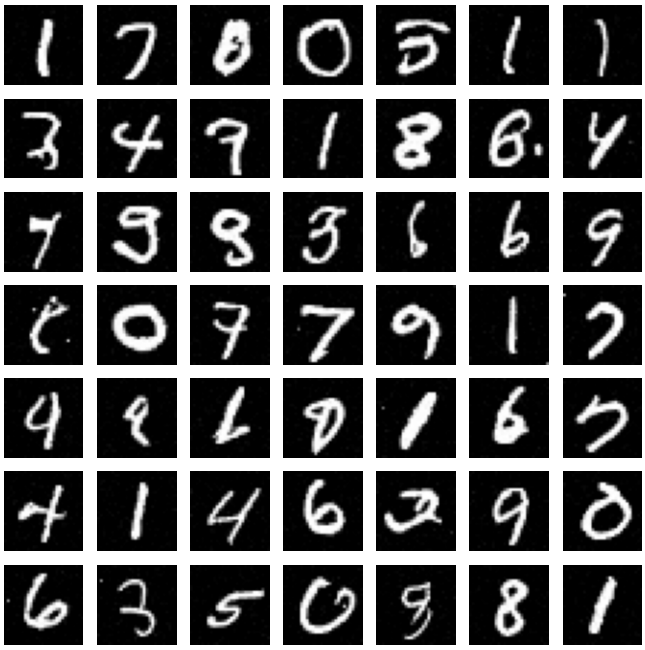


Figure B.15.: Generated MNIST samples by the implicit hollow Transformer trained with L_{CTEcrm}, using analytical sampling with a step size of $\tau = 0.00067$.

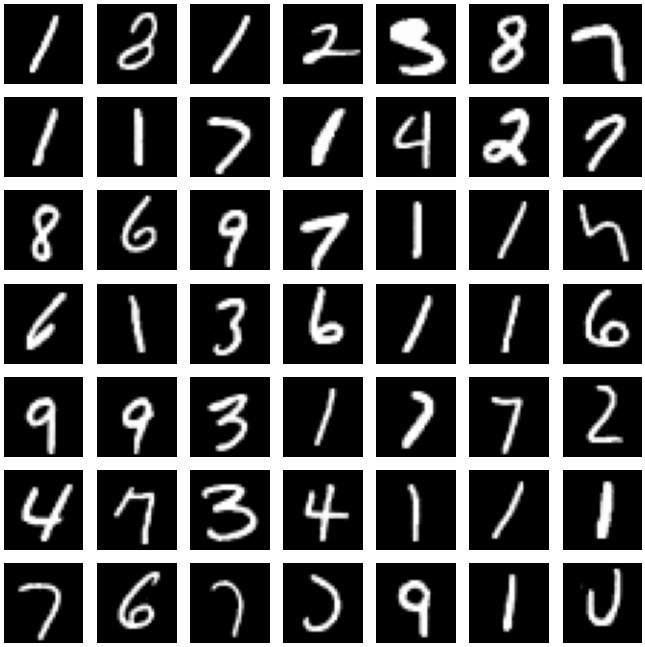


Figure B.16.: Generated MNIST samples by the implicit hollow Transformer trained with L_{CRMll}, using analytical sampling with a step size of $\tau = 0.00067$.

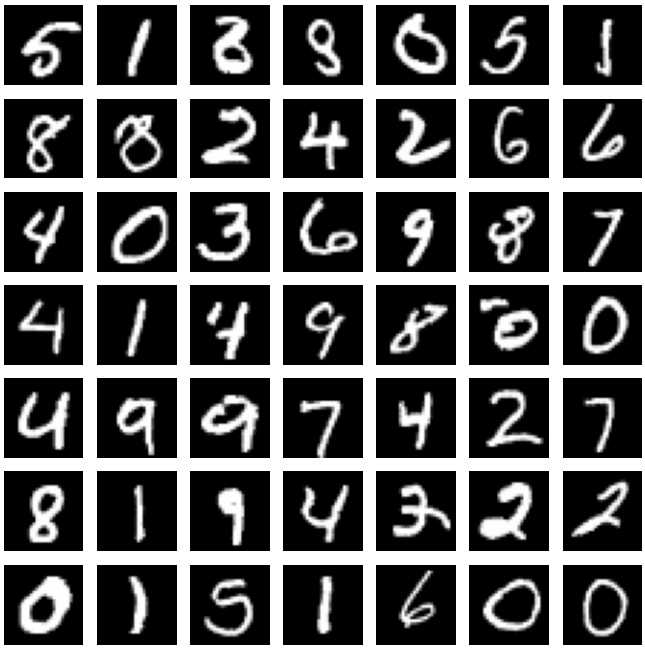


Figure B.17.: Generated MNIST samples by the τLDR model trained with L_{ll}, using midpoint tau-leaping with a step size of $\tau = 0.00067$.

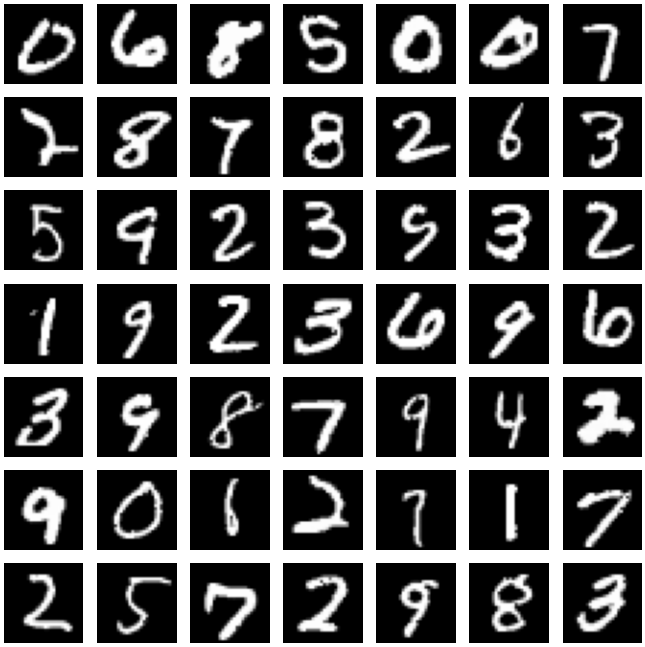


Figure B.18.: Generated mnist samples by the τLDR model trained with L_{CTEll}, using midpoint tau-leaping with a step size of $\tau = 0.00067$.

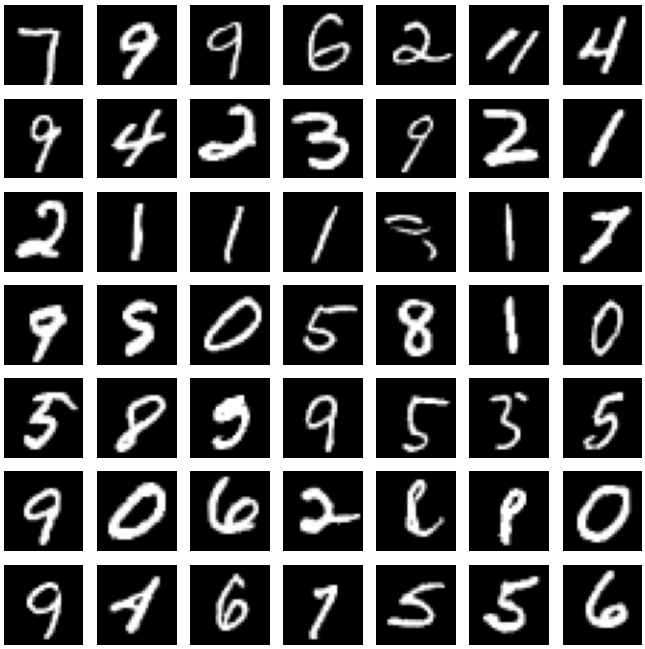


Figure B.19.: Generated mnist samples by the D3PM model trained with L_{DTEll}.