Stochastic Heat Diffusion Models

Marc T. Law **Xiaoyu Wang** **Lucas Berry** **Anthony L. Caterini** **Gabriel Loaiza-Ganem**

Layer 6 AI

Abstract

This work introduces a family of generative diffusion models based on the stochastic heat equation. The deterministic heat equation has natural connections with kernels over discrete structures and has been successfully applied in machine learning to discriminative tasks for various types of data such as graphs. We extend it to a stochastic version by formulating our diffusion as a linear diffusion whose forward transition kernels and clean-sample-conditioned reverse posteriors are Gaussian and available in closed form when conditioned on the Laplacian matrix. Our diffusion process propagates noise among neighbours under the guidance of a Laplacian matrix that describes the global structure of the input. Following an approach based on flow matching, we train a denoising neural network that takes noisy representations as input and gradually removes noise to generate clean samples. We experimentally show that, by exploiting the Laplacian during training, our decoder is able to recover the global structure of the input.

1 Introduction

Similarity-based methods form a fundamental class of algorithms in machine learning, operating on the principle that similar data samples should be treated in the same manner. The core idea involves defining some measure of similarity between pairs of samples, which is often represented as a (kernel) similarity matrix, a (possibly weighted) adjacency matrix, or a Laplacian matrix. The chosen matrix then becomes the foundation for various tasks such as clustering [38, 53], classification [10], and dimensionality reduction [5]. In this paper, we focus on the Laplacian matrix. Through its connection with the Laplace–Beltrami operator [5, 26, 29], the Laplacian encodes geometric relationships among data points assumed to lie on a Riemannian manifold [44]. For some types of discrete structures such as graphs, it is common to use a Laplacian matrix to simulate message passing through the notion of heat transfer between node representations.

We propose to extend heat transfer methods to the context of generative diffusion models by guiding a Wiener process through a given Laplacian matrix in order to propagate noise among neighbouring or similar samples. Through the connection with the Laplace-Beltrami operator [44], our method can be interpreted as diffusing noise along the underlying manifold instead of considering the noise to be diffused element-wise independently. We propose different ways to train a denoising decoder (or *denoiser* for short) to generate clean samples when given noisy data as input, and we experimentally show that our method recovers the global structure of the data.

Exploiting the Laplacian matrix has been proposed for image generation in contexts where the eigenvectors of the Laplacian matrix are given both during training and inference [20], which simplifies the training process and makes it similar to standard diffusion approaches. It has also been used for graph generation but only to reconstruct the adjacency matrix of the graph [28, 36, 48], ignoring the generation of nodes. Our method can be seen as an extension of flow matching [33] in which a Laplacian matrix is used to construct training paths and targets, but is not an input to the neural network and is not required during inference.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

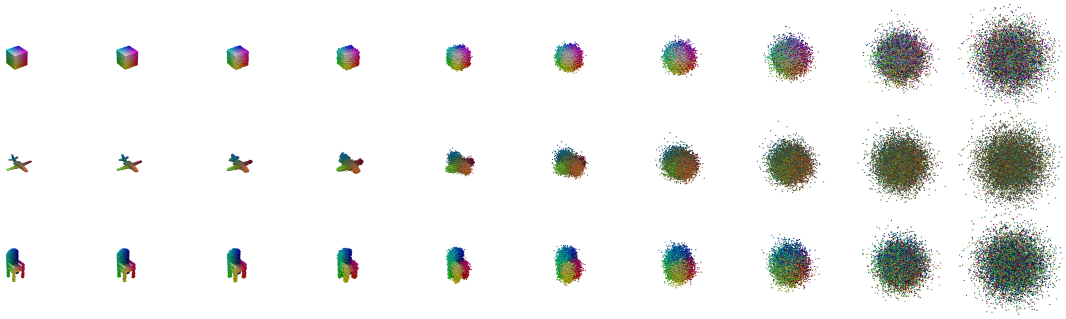


Figure 1: Illustration of stochastic-heat conditional trajectories on 3-dimensional point clouds, using standard-normal initialisation as a finite-time approximation.

2 Heat transfer on manifolds and graphs

This section introduces our notation and briefly explains heat transfer on Riemannian manifolds and graphs. We refer the reader to [5, 9, 26, 44] for details. Our approach is inspired by this literature, and Section 3.1 introduces stochasticity through a Wiener process.

Graphs are general structures that can represent different types of data, such as images for which the neighbours of each pixel node can be identified spatially. In general, node neighbourhoods can be difficult to define, and their structure may vary greatly from node to node. For tasks such as clustering [38, 53] or dimensionality reduction [5, 29], prior work relates graph Laplacians to the Laplace-Beltrami operator, thereby encoding manifold geometry in discrete node relationships.

Notation. We denote the identity matrix by $\mathbf{I}$, and the all-ones vector by $\mathbf{1}$. We consider n items that are either points lying on a connected Riemannian manifold (e.g., $\mathbb{R}^d$) or nodes of a given undirected, possibly disconnected graph. In either case, we associate the items with an undirected graph $\mathcal{G} = (V, E)$. The graph has n nodes $V = \{v_i\}_{i=1}^n$ and edges $E \subseteq V \times V$. Each node v_i contains a d-dimensional feature vector $\mathbf{n}_i$ that represents some attributes of the node. For instance, these attributes could represent the 3D coordinates of the node in the context of 3D point clouds (see Fig. 1). The (possibly weighted) symmetric adjacency matrix $\mathbf{A} \in [0, 1]^{n \times n}$ satisfies $\mathbf{A}_{ij} > 0$ if $(v_i, v_j) \in E$, and $\mathbf{A}_{ij} = 0$ otherwise. For instance, one can define $\mathbf{A}_{ij} = \exp(-\|\mathbf{n}_i - \mathbf{n}_j\|_2^2)$ if one wants the adjacency information to be induced by the standard Riemannian structure on $\mathbb{R}^d$ (see Appendix A for details). $\mathbf{A}$ can also be defined in a non-smooth manner (e.g., $\mathbf{A}_{ij} = 1$ if the distance between $\mathbf{n}_i$ and $\mathbf{n}_j$ is below a threshold, and $\mathbf{A}_{ij} = 0$ otherwise). The diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ is defined so that $\forall i, \mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. We define $\mathbf{L}_{\gamma,\delta} := \gamma \left(\frac{1}{n}(\mathbf{A} - \mathbf{D}) - \delta \mathbf{I}\right)$ as a shifted and scaled negative Laplacian of $\mathcal{G}$, where $\gamma > 0$ is a scaling hyperparameter and $-\gamma\delta\mathbf{I}$ is an absorption term. For simplicity, we refer to $\mathbf{L}_{\gamma,\delta}$ as the negative Laplacian throughout the paper. In the literature, δ is usually defined to be 0, which makes $\mathbf{L}_{\gamma,0}$ negative semidefinite with one eigenvalue equal to 0 when $\mathcal{G}$ is connected. In this paper, we set $\delta > 0$ to make $\mathbf{L}_{\gamma,\delta}$ negative definite and satisfy conditions that will ensure that our forward process tends to a standard normal distribution in Section 3. To simplify the notation, we write $\mathbf{L}$ instead of $\mathbf{L}_{\gamma,\delta}$.

Heat transfer. The d-dimensional node feature vectors can be stacked into a single matrix $\mathbf{N} \in \mathbb{R}^{n \times d}$, which represents some initial representation of the n nodes at time $t = 0$. In other words, we define $\mathbf{X}(t) \in \mathbb{R}^{n \times d}$ as the node representations at time $t \geq 0$, such that $\mathbf{X}(0) = \mathbf{N}$ is the initial condition of the heat equation defined as $\forall t \geq 0, \mathrm{d}\mathbf{X}(t) = \mathbf{L}\mathbf{X}(t)\mathrm{d}t$. When $\mathbf{L}$ is constant over time, the solution of the heat equation is $\forall t \geq 0, \mathbf{X}(t) = e^{t\mathbf{L}}\mathbf{X}(0)$, where $e^{t\mathbf{L}}$ denotes the matrix exponential of $t\mathbf{L}$. This solution can be interpreted as a continuous way of performing message passing in the graph $\mathcal{G}$ for continuous time t by exploiting the distance or neighbourhood information included in the Laplacian matrix. Graph heat transfer is a standard technique in spectral graph theory, and it has been used in different applications such as dimensionality reduction via Laplacian eigenmaps [5], 3D mesh surface classification or segmentation [45], or generalisations of graph neural networks [6]. Like these approaches, we consider node representations that are perturbed over time by the (matrix exponential of the negative of the) Laplacian matrix of the graph. However, we introduce an efficient way to inject noise into those representations via a Wiener process, so that we can sample noisy examples in closed form.

3 Noising encoder or forward diffusion process

Like most diffusion models, our framework can be seen as a denoising autoencoder [34, 52]. Its noising encoder does not require a neural network and samples noisy node representations in closed form. We describe here the forward process of our framework. We refer the reader to Examples 9.2.12 and 9.3.4 of [4], Chapter 4.8 of [25], and Appendix B for details.

3.1 General formulation of our linear Stochastic Differential Equation (SDE)

Given $\mathbf{X}(0)$ and a matrix $\mathbf{\Delta} \in \mathbb{R}^{n \times n}$, our encoder produces noisy representations $\mathbf{X}(t) \in \mathbb{R}^{n \times d}$ for $t > 0$. The forward process that we describe below is applied d times (once for each column of $\mathbf{X}(t)$). We therefore simplify our notation by writing $\mathbf{x}(t) \in \mathbb{R}^n$ or $\mathbf{x}_t \in \mathbb{R}^n$ interchangeably to describe any column of $\mathbf{X}(t)$. More precisely, we describe our diffusion process based on an n-dimensional Wiener process $\mathbf{w} = \{\mathbf{w}(t) : t \geq 0\}$ with components $w_1(t), \ldots, w_n(t)$, which are n independent scalar Wiener processes. To obtain an $n \times d$ matrix-valued Wiener process, the n-dimensional process described below can be repeated by using d independent copies of $\mathbf{w}$ (one for each column of $\mathbf{X}(t)$). We formulate our noising encoder as the following linear SDE on each $\mathbf{x}(t)$:

$$\forall t \geq 0, \quad \mathrm{d}\mathbf{x}(t) = (\mathbf{\Delta}\mathbf{x}(t) + \mathbf{a})\,\mathrm{d}t + \mathbf{B}\,\mathrm{d}\mathbf{w}(t) \tag{1}$$

where $\mathbf{\Delta}$ and $\mathbf{B}$ are $n \times n$ symmetric matrices that commute (i.e., $\mathbf{B}\mathbf{\Delta} = \mathbf{\Delta}\mathbf{B}$), $\mathbf{\Delta}$ is nonsingular, and $\mathbf{a}$ is a constant n-dimensional vector. Eq. (1) is known as the Ornstein–Uhlenbeck process when $n = 1$ and $\mathbf{\Delta} < 0$. Eq. (1) is reformulated as the following stochastic integral equation:

$$\forall u \geq s \geq 0, \quad \mathbf{x}(u) = \mathbf{x}(s) + \int_s^u (\mathbf{\Delta}\mathbf{x}(t) + \mathbf{a})\,\mathrm{d}t + \int_s^u \mathbf{B}\,\mathrm{d}\mathbf{w}(t) \tag{2}$$

where the first integral is a Lebesgue integral, and the second integral is an Itô stochastic integral. Eq. (2) can be approximated by means of Itô sums converging in mean square as explained in Appendix B. However, to be accurate, such an approximation of $\mathbf{x}(u)$ might be expensive to compute in practice due to the large number of terms in the sum. Instead, we exploit the fact that our diffusion process is a linear diffusion with Gaussian transition kernels when conditioned on a fixed $\mathbf{\Delta}$. These kernels have simple closed-form expressions for any value of $u \geq 0$, provided that (i) $\mathbf{\Delta}$ is nonsingular and (ii) we are given an earlier state $\mathbf{x}(s)$ such that $0 \leq s \leq u$. When conditioned on $\mathbf{\Delta}$, the process is Markov in the sense that if we are given a set of ordered states $\mathbf{x}(t_0), \mathbf{x}(t_1), \ldots, \mathbf{x}(s)$ where $t_0 < t_1 < \cdots < s$, then any later state $\mathbf{x}(u)$ where $u \geq s$ depends only on $\mathbf{x}(s)$. To simplify the notation, we consider that $t_0 = 0$ is the initial step of a sequence, t_T is the last step, and $t_i < t_j$ iff $i < j$.

3.2 Sampling noisy representations

We consider in this paper that we have $\mathbf{\Delta} = \mathbf{L}$ in Eq. (1); i.e., the negative of the Laplacian matrix of a graph $\mathcal{G}$ or of some set of n points in $\mathbf{X}(0) \in \mathbb{R}^{n \times d}$. In this paper, we consider that $(\mathbf{X}(0), \mathbf{L})$ is given as a pair, i.e., $\mathbf{L}$ can change depending on the starting condition. We choose the hyperparameter δ to be positive to make $\mathbf{L}$ nonsingular and symmetric negative definite, thereby ensuring the stability of our diffusion process. As explained below, the later choices $\mathbf{a} = \mathbf{0}$ and $\mathbf{B} = (-2\mathbf{L})^{1/2}$ ensure that the distribution tends to a standard normal distribution as t increases. If all the above conditions hold, then we can write in closed form the mean vector and covariance matrix of our normal distribution $q(\mathbf{x}(s+t)|\mathbf{x}(s), \mathbf{L})$ as follows (see Example 9.2.12 of [4] for details):

$$\forall s \geq 0, t \geq 0, \boldsymbol{\mu}_{s+t}(s, \mathbf{x}, \mathbf{L}) = e^{t\mathbf{L}}\,(\mathbf{x}(s) - \boldsymbol{\beta}) + \boldsymbol{\beta} \text{ where } \boldsymbol{\beta} := -\mathbf{L}^{-1}\mathbf{a}, \tag{3}$$

$$\forall s \geq 0, t \geq 0, \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \mathbf{B}(2\mathbf{L})^{-1}(e^{2t\mathbf{L}} - \mathbf{I})\mathbf{B}. \tag{4}$$

Formally, given some matrix $\mathbf{L}$ and some prior state $\mathbf{x}(s)$ at time $s \geq 0$, $\boldsymbol{\mu}_{s+t}(s, \mathbf{x}, \mathbf{L})$ and $\mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L})$ are the mean vector and covariance matrix at time $s + t$, respectively. Since we have defined $\mathbf{L}$ to be symmetric negative definite, we have $\lim_{t \to +\infty} e^{t\mathbf{L}} = \mathbf{0}$. To satisfy $\lim_{t \to +\infty} \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \mathbf{I}$, we set $\mathbf{B} = (-2\mathbf{L})^{1/2}$:

$$\mathbf{B} = (-2\mathbf{L})^{1/2} \implies \forall s \geq 0, t \geq 0, \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \mathbf{I} - e^{2t\mathbf{L}}. \text{ (See details in Appendix B.2)} \tag{5}$$

Eq. (5) and Eq. (3) imply $\lim_{t \to +\infty} \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \mathbf{I}$ and $\lim_{t \to +\infty} \boldsymbol{\mu}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \boldsymbol{\beta}$, where $\boldsymbol{\beta} := -\mathbf{L}^{-1}\mathbf{a}$. If $\mathbf{a} = \mathbf{0}$ and either t_T or γ (i.e., the scale of $\mathbf{L}_{\gamma,\delta}$) is large, we have the following

approximations: $\boldsymbol{\mu}_{t_T}(0, \mathbf{x}, \mathbf{L}) \approx \mathbf{0}$ and $\boldsymbol{\Sigma}_{t_T}(0, \mathbf{x}, \mathbf{L}) \approx \mathbf{I}$. In practice, we use $\mathbf{x}(t_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as a finite-time approximation, so every element of $\mathbf{x}(t_T)$ is sampled independently from $\mathcal{N}(0, 1)$. In this paper, we set $\mathbf{a} = \mathbf{0}$, but other formulations of $\mathbf{a}$ are possible.

Sampling. We can draw a noisy sample $\mathbf{x}(s + t)$ directly from a multivariate normal distribution $q(\mathbf{x}(s + t)|\mathbf{x}(s), \mathbf{L}) := \mathcal{N}(\boldsymbol{\mu}_{s+t}(s, \mathbf{x}, \mathbf{L}), \boldsymbol{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}))$ for any step size $t \geq 0$ and any initial step $s \geq 0$ when given $\mathbf{x}(s)$ and $\mathbf{L}$. For instance, in Fig. 1, the initial state $\mathbf{x}(t_0) \in [-1, 1]^n$ corresponds to one coordinate of a three-dimensional point cloud, and the Laplacian is constructed based on a Radial Basis Function (RBF) kernel as explained in Appendix A. Each sample $\mathbf{x}(t_{i+1})$ can be drawn from $\mathcal{N}(e^{(t_{i+1}-t_i)\mathbf{L}}\mathbf{x}(t_i), \mathbf{I} - e^{2(t_{i+1}-t_i)\mathbf{L}})$ for $i \in \{0, \dots, T-1\}$.

Diffusion process. As explained in Example 9.3.4 of [4] and page 152 of [25], the solution of our autonomous linear equation is a time-homogeneous diffusion process, which means that the transition probability of the process depends only on its duration and the last prior state of $\mathbf{x}$. Its drift is $b(t, \mathbf{x}) = \mathbf{L}\mathbf{x} + \mathbf{a}$, its diffusion coefficient is $\sigma(t, \mathbf{x}) = \mathbf{B} = (-2\mathbf{L})^{1/2}$, and its diffusion matrix is $\sigma(t, \mathbf{x})\sigma(t, \mathbf{x})^\top = \mathbf{B}\mathbf{B}^\top = -2\mathbf{L}$. Moreover, Eq. (1) has a pathwise unique strong solution $\mathbf{x}(t)$ on $[t_0, t_T]$ with $\sup_{t\in[t_0,t_T]} \mathbb{E}(\|\mathbf{x}(t)\|_2^2) < \infty$.

Practical considerations. It is worth noting that we never need to calculate the matrix $\mathbf{B}$. Our forward process only relies on the matrix exponential $e^{t\mathbf{L}}$ (and $e^{2t\mathbf{L}} = e^{t\mathbf{L}}e^{t\mathbf{L}}$) for some step size $t > 0$ when calculating the mean vector and covariance matrix in Eq. (3) and Eq. (5). A naïve approach would have algorithmic complexity $\mathcal{O}(n^3)$ when calculating $e^{t\mathbf{L}}$ each time a noisy representation is sampled. In practice, we may preprocess the eigendecomposition of $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\top$ offline to improve scalability. Since $\boldsymbol{\Lambda}$ is diagonal, the matrix exponential $e^{t\boldsymbol{\Lambda}}$ is also a diagonal matrix and is thus $\mathcal{O}(n)$ to compute; we can then write $e^{t\mathbf{L}} = \mathbf{U}e^{t\boldsymbol{\Lambda}}\mathbf{U}^\top$ for all t. Similarly, for the covariance matrix, let us define a vector $\boldsymbol{\omega} := \text{diag}(e^{2t\boldsymbol{\Lambda}})$. The covariance matrix can be written as $\mathbf{I} - e^{2t\mathbf{L}} = \mathbf{U}\left(\text{diag}(\mathbf{1} - \boldsymbol{\omega})\right)\mathbf{U}^\top$. In order to sample efficiently from any normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, having $\boldsymbol{\mu}$ and some matrix $\mathbf{C}$ that satisfies $\boldsymbol{\Sigma} = \mathbf{C}\mathbf{C}^\top$ is sufficient; we formulate $\mathbf{C} = \mathbf{U}\left(\text{diag}(\mathbf{1} - \boldsymbol{\omega})^{1/2}\right)$. Note that we also exploit $\mathbf{U}$ and $\boldsymbol{\Lambda}$ to efficiently compute the *velocity field* and *average velocity* in the section below.

4 Denoising decoder or reverse-time process

We present two approaches based on conditional flow matching [33] to train a denoiser neural network that generates clean samples. Our forward process is a linear SDE with known drift and diffusion matrix, so the reverse-time process can be formulated as a reverse-time SDE. However, the reverse SDE depends on $\mathbf{L}$, which is not given at inference time. Details are given in Appendix C.1.

4.1 Flow matching

One goal of flow matching [33] is to train a denoiser that simulates an ordinary differential equation (ODE) instead of an SDE. We explain how to adapt flow matching to our framework; we refer the reader to Appendix C.3 for details. We reverse the time parameter at the level of conditional marginals: for every $t \in [0, 1]$, the target marginal of $\bar{\mathbf{x}}_t$ is the conditional law of $\mathbf{x}(t_T(1 - t))$ given the clean sample and its Laplacian. Hereafter, $\bar{\mathbf{x}}_t$ denotes a flow-matching ODE coupling of these marginals, not the pathwise time reversal of the forward SDE. With this formulation, $\bar{\mathbf{x}}_1 = \mathbf{x}(0)$ is the original clean sample, while $\bar{\mathbf{x}}_0$ has the noisy marginal at time t_T. To simplify the notation, we set the scale of $\mathbf{L}$ to γt_T instead of γ, which can be seen as redefining $\mathbf{L} \leftarrow t_T\mathbf{L}$. Under the exact finite-time conditional process, the time-dependent mean vector and covariance matrix are written for all $t \in [0, 1]$ as $\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) := e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1$ and $\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) := \mathbf{I} - e^{2(1-t)\mathbf{L}}$.

Our conditional velocity field $\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) := \frac{1}{2}\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}^{-1}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}_t - \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)) + \bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)$ is:

$$\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) = \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1. \tag{6}$$

Although the velocity field is undefined at $t = 1$, applying the left-endpoint Euler method avoids evaluating the endpoint itself. However, the velocity field remains unbounded as $t \to 1^-$, which makes the ODE stiff near the endpoint and can cause numerical errors. This behaviour motivates the nonuniform scheduler and the formulation of the average velocity in Section 4.2. By definition, Eq. (6) satisfies $\forall a \in [0, 1), \forall b \in [a, 1], \bar{\mathbf{x}}_b = \bar{\mathbf{x}}_a + \int_a^b \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})\mathrm{d}t$. A neural network $\text{net}_\theta(\bar{\mathbf{x}}_t, t)$ can be trained to predict the conditional instantaneous velocity in Eq. (6) as explained in Appendix C.3.

Algorithm 1 Training of the average velocity prediction algorithm

Require: hyperparameters $\eta > 0$, $p \in [0, 1]$, and $\delta > 0$ to make $\mathbf{L}$ symmetric negative definite
while Not converged **do**
Sample clean training example containing $\mathbf{x}(0) = \bar{\mathbf{x}}_1$ with its Laplacian matrix $\mathbf{L}$
Sample a and b such that $0 \le a < b \le 1$, draw $\bar{\mathbf{x}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and calculate $\bar{\mathbf{x}}_a$ (see Eq. (7))
Calculate the conditional *average velocity* $\mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L}) = \frac{1}{b-a}(\bar{\mathbf{x}}_b - \bar{\mathbf{x}}_a)$ (see Eq. (7))
Minimise $\texttt{stop_gradient}(1/(\|\mathbf{q}\|_2^2 + \eta)^p)\|\mathbf{q}\|_2^2$ where $\mathbf{q} := \mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L}) - \mathrm{net}_\theta(\bar{\mathbf{x}}_a, a, b)$
end while
Return net_θ

Algorithm 2 Inference of the average velocity prediction algorithm

Require: Time scheduler of T steps: $0 = \tau_0 < \cdots < \tau_T = 1$ s.t. $t_{T-i} = t_T(1 - \tau_i)$ (see Eq. (8))
Draw $\bar{\mathbf{x}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $j = 1$ to T **do**
Define $\bar{\mathbf{x}}_{\tau_j} = \bar{\mathbf{x}}_{\tau_{j-1}} + (\tau_j - \tau_{j-1})\,\mathrm{net}_\theta(\bar{\mathbf{x}}_{\tau_{j-1}}, \tau_{j-1}, \tau_j)$
end for
Return clean sample $\bar{\mathbf{x}}_1$

4.2 Average velocity prediction

Instead of training a (neural network) model to predict the conditional instantaneous velocity field $\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})$ at time t, we train a model $\mathrm{net}_\theta(\bar{\mathbf{x}}_a, a, b)$ that tries to predict conditional average velocities over intervals $0 \le a < b \le 1$. Because the network receives $(\bar{\mathbf{x}}_a, a, b)$ but not $(\bar{\mathbf{x}}_1, \mathbf{L})$, its population regression output is the marginal average velocity conditioned on $(\bar{\mathbf{x}}_a, a, b)$; this output is used to estimate the less noisy representation $\bar{\mathbf{x}}_b$.

First, we note that $\mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L}) := \frac{1}{b-a}\int_a^b \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})\mathrm{d}t = \frac{1}{b-a}(\bar{\mathbf{x}}_b - \bar{\mathbf{x}}_a)$ is called the *average velocity* conditioned on $(\bar{\mathbf{x}}_1, \mathbf{L})$ from a to b [16]. As the number of steps increases and the intervals $[a, b]$ get smaller, $\mathbf{v}_{a,b}$ approaches $\mathbf{u}_a(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L})$, making our algorithm similar to standard flow matching. We show in Appendix C.4 that the conditional trajectory $\bar{\mathbf{x}}_t$ can be written in closed form when given some clean sample $(\bar{\mathbf{x}}_1, \mathbf{L})$ and initial condition $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$:

$$\forall t \in [0, 1],\ \bar{\mathbf{x}}_t = e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 + (\mathbf{I} - e^{2(1-t)\mathbf{L}})^{1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1). \tag{7}$$

We can then train our neural network to predict either the direct target $\bar{\mathbf{x}}_b$ or the conditional average velocity $\mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L})$. The corresponding training and inference algorithms for the latter are illustrated in Algorithms 1 and 2, respectively. Our training algorithm is similar to MeanFlow [16], except that we use a closed-form expression for the conditional average velocity as the target instead of an approximation. The theoretical conditional distribution $\mathcal{N}(e^{\mathbf{L}}\bar{\mathbf{x}}_1, \mathbf{I} - e^{2\mathbf{L}})$ of $\bar{\mathbf{x}}_0$ induced by the finite-time process generally differs from the standard normal distribution used in practice. For simplicity, especially at inference time, we draw $\bar{\mathbf{x}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ during both training and inference and treat the resulting trajectories as finite-time approximations to the exact stochastic-heat conditional marginals.

Inference and noise scheduling. We observed that using multiple steps $0 = \tau_0 < \cdots < \tau_T = 1$ produces better results (see Table 3 for analysis). Under our chosen large γ scaling, the norm of the velocity field is typically small near $t = 0$ and diverges almost surely as $t \to 1^-$ when $\bar{\mathbf{x}}_0$ is sampled from a Gaussian distribution. We therefore use the following exponential noise scheduler (see Fig. 2 and Appendix C.4.4):

$$\tau_i := \frac{1 - e^{-\frac{i\kappa}{T}}}{1 - e^{-\kappa}}, \tag{8}$$

where $\kappa > 0$ is a hyperparameter that is dataset-dependent. The choice of the noise scheduler is a challenging task that can have a significant impact on the performance of the decoder, and the best scheduling scheme when training diffusion models depends on the

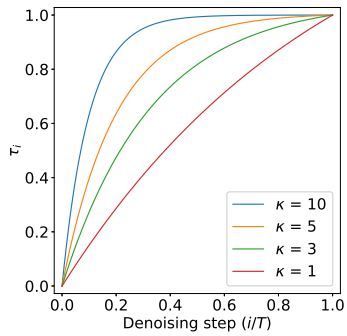


Figure 2: τ_i as a function of i/T.

task and the dataset [8, 32]. We observed that $\kappa \in \{1, 3, 5, 10\}$ works well when the original data points are in the cube $[-1, 1]^d$. Under the exact analytic conditional path, the distribution at time τ_i has covariance matrix $\bar{\mathbf{\Sigma}}_{\tau_i,\mathbf{L}}(\bar{\mathbf{x}}_1) := \mathbf{I} - e^{2(1-\tau_i)\mathbf{L}}$. When $\bar{\mathbf{x}}_0$ is instead drawn from a standard normal distribution, we use this covariance and the associated conditional marginals as finite-time approximations. Let $\bar{\mathbf{X}}_t \in \mathbb{R}^{n\times d}$ denote the matrix containing $\bar{\mathbf{x}}_t \in \mathbb{R}^n$ as one of its d columns. Fig. 1 illustrates three conditional trajectories using the finite-time approximation, all starting from the same noisy representation $\bar{\mathbf{X}}_0 \in \mathbb{R}^{n\times 3}$ with columns $\bar{\mathbf{x}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and ending at three different point clouds $\bar{\mathbf{X}}_1 \in [-1, 1]^{n\times 3}$. The trajectories use our closed-form average velocity and an exponential scheduler with $\kappa = 5$. For each point cloud, we calculate a different Laplacian from the pairwise RBF-kernel values of the corresponding clean point cloud $\bar{\mathbf{X}}_1$, as described in Section 2.

Loss metrics. Since our neural network is trained to predict the conditional *average velocity*, many techniques used in MeanFlow [16] can be adapted to our framework. In particular, if we define $\mathbf{q} := \mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L}) - \mathrm{net}_\theta(\bar{\mathbf{x}}_a, a, b)$, we use the gradient-reweighted surrogate $\texttt{stop_gradient}(1/(\|\mathbf{q}\|_2^2+\eta)^p)\|\mathbf{q}\|_2^2$, where $\eta > 0$ is a small positive stabiliser. Letting $r := \|\mathbf{q}\|_2^2$, its parameter gradient is $(r+\eta)^{-p}\nabla r$. For $0 \le p < 1$, this is exactly the gradient of $(r+\eta)^{1-p}/(1-p)$; as $\eta \to 0$, this objective is proportional to $\|\mathbf{q}\|_2^{2(1-p)}$. For $p = 1$, the surrogate gradient is exactly the gradient of $\log(r + \eta)$. We tune p within the interval $[0, 1]$.

Reference guidance. Let P_{target} and P_{ref} denote the joint populations of clean examples, their Laplacians, and their conditional trajectories for the target and reference distributions, respectively. For $P \in \{P_{\mathrm{target}}, P_{\mathrm{ref}}\}$, we define the corresponding marginal average velocity at a query state $\mathbf{z}$ by

$$\mathbf{v}^P_{a,b}(\mathbf{z}) := \mathbb{E}_P[\mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L}) \mid \bar{\mathbf{x}}_a = \mathbf{z}]\,. \tag{9}$$

We then define the reference-guided (RG) marginal average velocity as

$$\mathbf{v}^{\mathrm{RG}}_{a,b}(\mathbf{z}) := \omega\mathbf{v}^{P_{\mathrm{target}}}_{a,b}(\mathbf{z}) + (1-\omega)\mathbf{v}^{P_{\mathrm{ref}}}_{a,b}(\mathbf{z}), \tag{10}$$

where $\omega \ge 0$ is an arbitrary hyperparameter. The closed-form conditional average velocity for a fixed pair $(\bar{\mathbf{x}}_1, \mathbf{L})$ does not depend on a population label. Reference Guidance (RG) arises only after taking the two population expectations in Eq. (9). In Algorithm 1, these expectations are estimated using samples from their respective populations, and the training target $\mathbf{v}_{a,b}(\bar{\mathbf{x}}_a|\bar{\mathbf{x}}_1, \mathbf{L})$ is replaced by a corresponding estimate of $\mathbf{v}^{\mathrm{RG}}_{a,b}(\mathbf{z})$. We typically set $\omega \in (1, 5]$.

5 Covering the data distribution

We evaluate SHDM on the two-dimensional synthetic experiment introduced by [23] to motivate model guidance. One criticism of standard diffusion models is that their generated (clean) samples may fall outside high-density regions of the training distribution. Guidance has been proposed to suppress low-density outliers by steering sampling toward well-learned high-density regions. We test whether SHDM reduces such outliers without guidance.

Dataset. The dataset introduced by [23] consists of 2-dimensional distributions similar to fractals, with our target distribution (i.e., ground truth) supported on the orange region in Fig. 3. Karras et al. [23] argue that standard score matching leads to the generation of many outliers, which is typically not desirable. They then propose some model guidance to generate samples within the class distribution, despite not covering the whole region.

Proposed method. Since the points lie in the Euclidean space $\mathbb{R}^2$, we propose to exploit the connection between $\mathbb{R}^2$ and the Laplacian (see Appendix A). At each iteration, we sample n points $\{\mathbf{n}_i \in \mathbb{R}^2\}_{i=1}^n$ and formulate our weighted adjacency matrix such that $\mathbf{A}_{ii} = 0$ and $\mathbf{A}_{ij} = \exp(-10\|\mathbf{n}_i - \mathbf{n}_j\|^2)$ if $i \ne j$, and we formulate $\mathbf{L} = \frac{1}{n}(\mathbf{A} - \mathrm{diag}(\mathbf{A}\mathbf{1})) - \delta\mathbf{I}$ where $\delta = 10^{-5}$. Without requiring guidance, SHDM generates most samples in high-probability regions without substantially reducing diversity. This addresses the low-density outliers that motivated the introduction of guidance in [23].

Guidance. As explained in Section 4.2, our model can include reference guidance during training. For the target average velocity $\mathbf{v}^{P_{\mathrm{target}}}_{a,b}$, we sample $\bar{\mathbf{x}}_1$ directly from the target distribution without noise (i.e., the orange region). For the reference average velocity $\mathbf{v}^{P_{\mathrm{ref}}}_{a,b}$, we construct a reference population by adding zero-mean Gaussian noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ to each target sample $\bar{\mathbf{x}}_1$, where

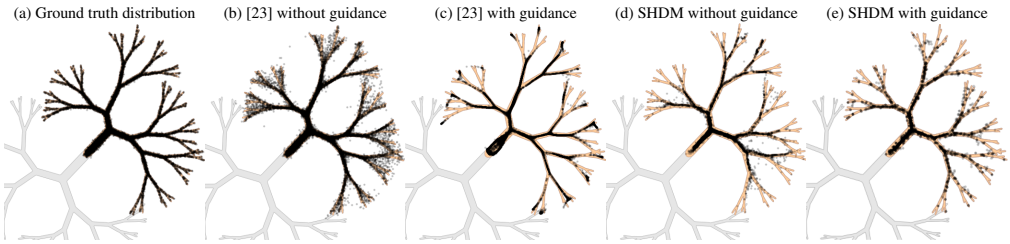


Figure 3: (a) Ground truth samples drawn from the orange class distribution. (b) Conditional sampling without guidance generates many outliers. (c) The method in [23] improves sampling in high-probability regions compared to no guidance. (d) Our method generates many samples within the orange region without reference guidance. (e) Samples generated by our method with reference guidance.

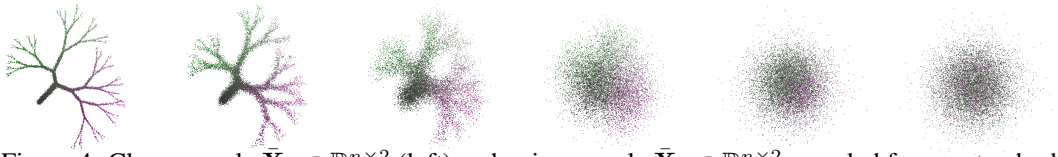


Figure 4: Clean sample $\bar{\mathbf{X}}_1 \in \mathbb{R}^{n\times 2}$ (left) and noisy sample $\bar{\mathbf{X}}_0 \in \mathbb{R}^{n\times 2}$ sampled from a standard normal distribution as a finite-time approximation (right). The intermediate representations lie on the corresponding trajectory in Eq. (7), using our exponential scheduler with $\kappa = 10$ and $T = 5$.

Table 1: MMD distances between the ground truth distribution and points sampled by [23] with or without guidance, or by SHDM with or without guidance. SHDM performs better in terms of MMD.

Method	Baseline without guidance	Baseline with guidance	SHDM without guidance	SHDM with guidance
MMD ($\sigma^2 = 1$)	1.42×10^{-1}	1.77×10^{-1}	3.70×10^{-2}	3.00×10^{-2}
MMD ($\sigma^2 = 100$)	9.92×10^{-3}	1.39×10^{-2}	2.15×10^{-3}	2.01×10^{-3}

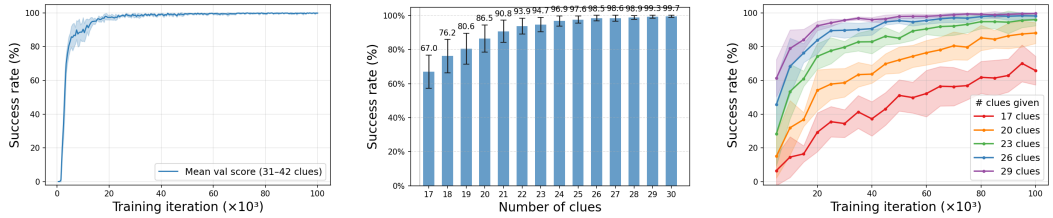
σ is small (we sample σ^2 uniformly in the interval $[0, 0.01)$ since we have $\bar{\mathbf{x}}_1 \in [-0.7, 1.5]^n$, and $\sigma = 0$ corresponds to no blurring). For this toy experiment, we set $\omega = 5$ in Eq. (10). In the reported implementation, we reuse the Laplacian computed from each clean target sample when evaluating its blurred reference sample. In practice, the Laplacian should instead be recomputed from each blurred sample so that it remains consistent with that reference sample.

Quantitative results. Table 1 reports the Maximum Mean Discrepancy (MMD) [17] between a fixed set of $5{,}000$ ground-truth samples and samples generated by the baseline variants in [23] (with and without guidance) and the SHDM variants (with and without reference guidance). MMD measures a distance between two probability distributions; we compute MMD using an RBF kernel with different bandwidth parameters σ^2. Although the baseline method without guidance samples many outliers, its MMD *w.r.t.* the ground truth distribution is slightly lower than the baseline with guidance. Nonetheless, SHDM achieves a lower MMD without reference guidance. SHDM with reference guidance performs slightly better than without reference guidance.

Qualitative comparison. Fig. 4 and Fig. 9 illustrate trajectories under SHDM's conditional velocity field, using Gaussian initialisation as a finite-time approximation. We visually compare our scheduler to linear and cosine schedulers as defined by [19, 39] when applied to the forward process of the standard DDPM in Appendix E.2. It is worth noting that these schedulers were proposed in the context of image generation, not for general types of structured data where similarities between points are available. This suggests that the global structure is better preserved by SHDM, which tends to keep similar points closer to each other when noise is added by exploiting the Laplacian.

6 Sudoku

A Sudoku can be expressed as a graph colouring problem where the goal is to construct a 9-colouring given a partial 9-colouring — a colour being an integer in this context. In its classic formulation, a node is one of the $n = 81$ cells, and it is connected to another node by an edge if they are in the same row, in the same column, or in the same 3×3 sub-grid. We formulate our node representations as a



(a) In-distribution evaluation on 31–42-clue Sudoku puzzles across training.

(b) OOD test on 17–30-clue Sudoku puzzles using the final checkpoint from each training run.

(c) Training continues to improve out-of-distribution generalisation.

Figure 6: Sudoku-solving results for in-distribution and out-of-distribution evaluations, averaged over five seeds.

matrix $\mathbf{X}(0) \in \{0,1\}^{n \times d}$ where $d = 9 \times 4 = 36$, and each node is represented by four 9-dimensional one-hot vectors that correspond to the index of the integer in the cell, the row, the column, and the 3×3 sub-grid of the cell. The 27 row, column, and sub-grid channels are held fixed as conditioning throughout the denoising process; only the nine digit channels are generated. Instead of considering the classic edge formulation (i.e., the conventional Sudoku constraint graph described above), we construct our adjacency matrix so that $\mathbf{A}_{ij} = 1$ if $i \neq j$ and the two cells represent the same integer (i.e., the same colour), and $\mathbf{A}_{ij} = 0$ otherwise. Moreover, we have $\mathbf{A}_{ii} = 0$ for every cell i. Although this construction yields a disconnected graph, choosing $\delta > 0$ makes $\mathbf{L}_{\gamma,\delta}$ negative definite, so connectedness is not required in practice.

Guidance. In addition to the fixed positional channels, the digit values of all clues are supplied as conditioning at every denoising step. The target population contains only valid Sudoku solutions consistent with the clues whereas the reference population may contain both valid and invalid solutions. To construct the reference population, we uniformly sample a permutation of the solution's digit values over the $m \leq n$ unknown cells while retaining the positional channels and clue values as conditioning.

8	3		1					2
	7		9	5		8		
								4
			3	6	1	7		
		1	7		9	5		8
					8		6	
	1		4	9				
	9				3	6		
2	8	3			7		9	

8	3	6	1	7	4	9	5	2
1	7	4	9	5	2	8	3	6
9	5	2	8	3	6	1	7	4
5	2	8	3	6	1	7	4	9
3	6	1	7	4	9	5	2	8
7	4	9	5	2	8	3	6	1
6	1	7	4	9	5	2	8	3
4	9	5	2	8	3	6	1	7
2	8	3	6	1	7	4	9	5

Figure 5: (left) Clues given as conditioning. (right) Valid solution generated by our model.

Inference. At inference time, our model generates a matrix in $\mathbb{R}^{n \times d}$ that represents a candidate solution, which is not necessarily a valid Sudoku solution. The fixed positional channels identify the 81 cells (see Fig. 5), and clue digit values remain available as conditioning throughout denoising. We then calculate the distance matrix between the generated digit representations and 81 one-hot vectors that represent the possible integers in the Sudoku square (i.e., nine copies of each of the nine digit one-hot vectors, see details in Appendix E.3). We assign each cell using a global assignment problem solver, such as the Hungarian algorithm [27]. This postprocessing enforces only that each digit occurs nine times globally; it does not enforce the row, column, or sub-grid constraints.

Experiment setup. In general, the fewer clues a Sudoku puzzle starts with, the harder it is to solve, with 17 clues widely considered the minimum for a uniquely solvable puzzle [37]. Following the training setup in related work [54, 55], we train our model with randomly sampled Sudoku puzzles[1] with anywhere from 31 to 42 given clues (inclusive). We test the trained model with both in-distribution data (i.e., clue-conditioned generation given 31 to 42 clues, inclusive) and harder out-of-distribution (OOD) data (i.e., given 17 to 30 clues, inclusive), and we evaluate the success rate of generating a valid solution. Only fully resolved puzzles are credited as successful in the evaluation.

Diffusion models as Sudoku solvers. Our model reaches a solution success rate of $99.9\% \pm 0.2\%$ on in-distribution data after 100,000 training iterations. Fig. 6a shows the learning curve, where the model converges rapidly in the first 30,000 iterations. On harder OOD data, the model still achieves a $91.4\% \pm 4.4\%$ success rate, demonstrating the strong generalisability of our heat diffusion approach. These results are reported as the mean $\pm$ sample standard deviation over five seeds. Fig. 6b further

[1]We use the Sudoku generator publicly available at `https://github.com/ankandrew/DeepSudoku`

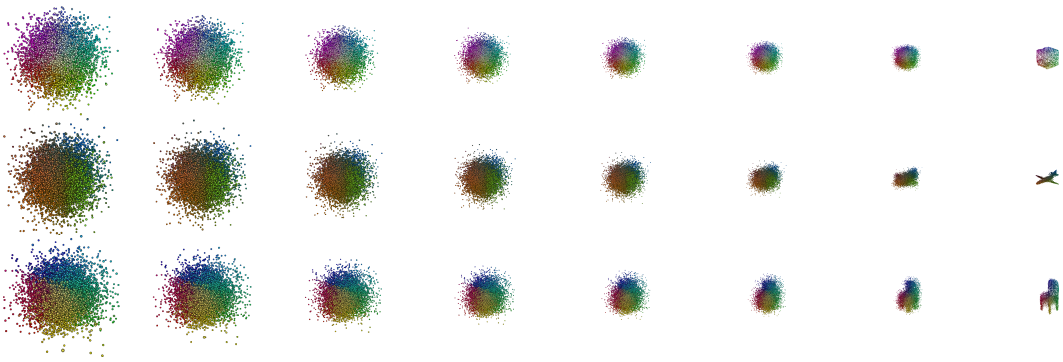


Figure 7: Illustration of the denoising process of SHDM for point clouds.

reports the solution success rate on puzzles with different numbers of initial clues (1,000 samples per test). Fig. 6c illustrates that even after convergence on the in-distribution data, the model's OOD generalisation performance continues to improve as training progresses (200 samples per test). Additional experimental details and analyses are provided in Appendix E.3.

7 Point clouds

We present a qualitative experiment on 3-dimensional point cloud generation, another task in which the node representations are continuous vectors in $\mathbb{R}^3$ corresponding to (x, y, z) coordinates. We consider various types of shapes, such as a cubic box, a chair, and an airplane from the ShapeNet [7] dataset (see examples in Fig. 1). At least 10,000 points are available per shape.

Subsampling of training points. Since our goal is to consider datasets that can potentially be very large, one natural question is whether or not our model is scalable and can learn the global structure if it sees only a smaller set of points at a time. To answer this question, we subsample $n = 10^3$ points at each iteration to form $\bar{\mathbf{X}}_1 \in [-1, 1]^{n\times 3}$, and we compute during training the corresponding matrix $\mathbf{L}$ and its eigendecomposition, which is very efficient on modern architectures when $n \leq 5{,}000$.

Guidance. Following Section 5, we estimate the target field $\mathbf{v}_{a,b}^{P_{\text{target}}}$ using clean samples drawn directly from the ground-truth point cloud. For the reference field $\mathbf{v}_{a,b}^{P_{\text{ref}}}$, we use a blurred reference population obtained by adding zero-mean Gaussian noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ to each clean point-cloud sample, where σ^2 is sampled uniformly from $[0, 0.01)$. We set $\omega = 5$ in Eq. (10), and $\kappa = 1$ in Eq. (8).

Fig. 7 illustrates some point clouds generated by our model when trained with 10^3 points per iteration and used to generate 8×10^3 points in a single shot at inference time. In this setup, we consider that our model is trained on a single shape at a time. Although the model had access only to partial observations during training, it generates point clouds similar to the target distributions. This suggests that our model supports mini-batch training for improved scalability.

8 Discussion and other applications

Limitations. As in many similarity-based approaches, one limitation of our method is that the similarity function between points must be defined to formulate the Laplacian. For instance, we chose a binary similarity based on the integer value in the Sudoku experiment, or an RBF kernel-based similarity in the other two tasks. Moreover, the hyperparameters $\gamma > 0$ and $\delta > 0$ of $\mathbf{L}_{\gamma,\delta}$ depend on the dataset and require tuning. A structure-free baseline is $\mathbf{L} = -\zeta\mathbf{I}$, where $\zeta > 0$ is a hyperparameter; this rescaled identity matrix avoids calculating an eigendecomposition during training. If feature blocks encode different similarity relations, each block can be diffused independently with its own Laplacian. The matrix $\mathbf{L}$ could also be learned. Another limitation is that we assume that $\mathbf{L}$ is constant over time. We leave the exploration of a matrix $\mathbf{L}$ that varies over time for future research. Lastly, learning the global structure might be difficult when n is large as $\mathbf{L}$ is an $n \times n$ matrix. Our experiment in Section 7 suggests that our model is able to learn the global data structure while being

trained in a mini-batch manner with a smaller number of points at a time. Another option would be to explore low-rank approximations of $\mathbf{L}$ to improve scalability.

Approximation of the average velocity. Our conditional average velocity algorithm should be viewed as an empirical multi-step approximation without an exact finite-step transport guarantee. As the number of denoising steps increases and the intervals get smaller, our approach becomes similar to standard flow matching because the conditional average velocity approaches the instantaneous conditional velocity. Calculating the marginal average velocity to be used as a training target is left for future work.

Graph generation. The above sections explain how to reconstruct real-valued node features $\mathbf{N}$. It is worth noting that it is possible to combine our approach with methods proposed by [28, 36, 48] that exploit the Laplacian to generate adjacency matrices (i.e., edges). Since our algorithm generates node features, generating graphs corresponds to predicting the existence of edges connecting pairs of generated nodes (or their edge features). It is also known in spectral graph theory that much of the structural information of a graph is contained in a subset of the eigenvectors of its Laplacian matrix. We call this set of eigenvectors $\mathbf{V} := [\mathbf{v}_1, \dots, \mathbf{v}_k] \in \mathbb{R}^{n \times k}$ where k is an arbitrary hyperparameter. The main idea is to stack some combination of $\mathbf{V}$ and $\mathbf{N}$ to form the initial condition of our problem (e.g., $\mathbf{X}(0) = [\mathbf{N}, \mathbf{V}]$ if we ignore the eigenvalues). SHDM is then applied to generate this combination, which is used to predict the existence or features of edges. We discuss such combinations in Appendix D, and we apply our graph generation framework to molecules on the QM9 dataset [41] in Appendix E.4. We show that our approach achieves competitive validity scores with relatively high novelty.

Blurring diffusion models. Hoogeboom & Salimans [20] proposed blurring diffusion models as an extension of the method introduced by Rissanen et al. [43]. Unlike Hoogeboom & Salimans [20], our forward process is written directly in matrix form and allows for more general parameters of a linear diffusion with Gaussian transition kernels conditional on $\mathbf{L}$, such as a non-zero value of $\boldsymbol{\beta}$ in Eq. (3), or a covariance matrix that allows $\mathbf{B} \neq (-2\mathbf{L})^{1/2}$ in Eq. (4). A major difference between their approach and ours is that Hoogeboom & Salimans [20] also assume that the matrix of eigenvectors $\mathbf{U}$ is given at inference time, which greatly simplifies both their training and inference algorithms that sample noisy representations in the same way as standard diffusion techniques used in DDPMs. In our context, $\mathbf{L}$ and its eigenvectors $\mathbf{U}$ are not given at inference time, so the reverse SDE cannot evaluate its diffusion matrix or its $\mathbf{L}$-conditioned score; see Appendices C.1 and C.2. To address this issue, we use flow matching, which avoids requiring either quantity at inference time and yields a closed-form expression of $\bar{\mathbf{x}}(t)$ for all $t \in [0, 1]$.

Positional encoding (PE). Since we consider unordered sets of nodes, our method does not require standard PE. When available, structural information or coordinates are still supplied as conditioning; this task-specific conditioning is distinct from adding standard positional encodings to the architecture. Our approach is designed for any node-permutation-equivariant architecture such as Transformers [50] without PE and Deep Sets [56].

Guidance. Reference guidance was helpful in the Sudoku experiment, whereas it did not significantly improve performance in the other experiments. We leave the exploration of other reference populations and guidance methods for future research.

9 Conclusion

We have proposed a diffusion model based on a stochastic version of the heat equation that exploits a given Laplacian matrix during training to define similarities between nodes. The similarity function can be defined in a continuous or discrete manner (e.g., based on kernel functions or on an adjacency matrix). We formulate our training algorithm as an approach based on flow matching. In particular, our model is trained to predict conditional *average velocity* targets, which we efficiently compute in closed form, and learns the corresponding marginal predictor from inference-time inputs. Our approach does not require access to the Laplacian at inference time, but can exploit it if available. We experimentally evaluate whether our approach preserves the Laplacian-defined structure across tasks that use different training-time similarity functions, including when the model is trained in a mini-batch setting. Our approach is general and can be extended to other types of data that can be formulated as graphs, such as images, time series, relational data, and neural networks.

Broader societal impact. Our work is mainly theoretical, so we see no direct negative societal impact.

Acknowledgments and Disclosure of Funding

We thank David Glukhov, Disen Liao, Nicholas Sharp, Tianchang Shen and Kevin Zhang for helpful discussions. We also thank the anonymous NeurIPS reviewers for feedback on an early version of this manuscript. As members of the NeurIPS 2026 Program Committee, Marc Law and Lucas Berry received complimentary full registrations for NeurIPS 2026.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL `https://arxiv.org/abs/1907.10902`.

[2] Nataraj Akkiraju, Herbert Edelsbrunner, Michael Facello, Ping Fu, EP Mucke, and Carlos Varela. Alpha shapes: definition and software. In *Proceedings of the 1st international computational geometry software workshop*, volume 63, 1995.

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

[4] Ludwig Arnold. *Stochastic Differential Equations: Theory and Applications*. Wiley, 1974. ISBN 9780471033592.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[6] Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1407–1418. PMLR, 18–24 Jul 2021.

[7] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] Ting Chen. On the importance of noise scheduling for diffusion models. *arXiv preprint arXiv:2301.10972*, 2023.

[9] Fan Chung and S-T Yau. Coverings, heat kernels and spanning trees. *the electronic journal of combinatorics*, pp. R12–R12, 1999.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[11] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 255–262. Omnipress; Madison, WI, USA, 2010.

[12] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[13] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[14] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4):551–559, 2003.

[15] Floor Eijkelboom, Grigory Bartosh, Christian A. Naesseth, Max Welling, and Jan-Willem van de Meent. Variational flow matching for graph generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=UahrHR5HQh.

[16] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=uWj4s7rMnR.

[17] 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.

[18] Ulrich G Haussmann and Etienne Pardoux. Time reversal of diffusions. *The Annals of Probability*, pp. 1188–1205, 1986.

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[20] Emiel Hoogeboom and Tim Salimans. Blurring diffusion models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=OjDkC57x5sz.

[21] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International conference on machine learning*, pp. 10362–10383. PMLR, 2022.

[22] Jaehyeong Jo, Dongki Kim, and Sung Ju Hwang. Graph generation with diffusion mixture. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22371–22405. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/jo24b.html.

[23] Tero Karras, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. Guiding a diffusion model with a bad version of itself. *Advances in Neural Information Processing Systems*, 37:52996–53021, 2024.

[24] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[25] Peter Eris Kloeden, Eckhard Platen, and Henri Schurz. *Numerical solution of SDE through computer experiments*. Springer Science & Business Media, 2012.

[26] Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pp. 315–322, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1558608737.

[27] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[28] Marc T. Law, Karsten Kreis, and Haggai Maron. Directed graph generation with heat kernels. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=60Gi1w6hte.

[29] Bruno Lévy. Laplace-beltrami eigenfunctions towards an algorithm that "understands" geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, pp. 13–13. IEEE, 2006.

[30] Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Q-UHqMorzil.

[31] Derek Lim, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ijK5hyxs0n.

[32] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common diffusion noise schedules and sample steps are flawed. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 5404–5411, 2024.

[33] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

[34] Gabriel Loaiza-Ganem, Brendan Leigh Ross, Rasa Hosseinzadeh, Anthony L Caterini, and Jesse C Cresswell. Deep generative models through the lens of the manifold hypothesis: A survey and new connections. *Transactions on Machine Learning Research*, 2024.

[35] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in neural information processing systems*, 31, 2018.

[36] Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. In *International Conference on Machine Learning*, pp. 15159–15179. PMLR, 2022.

[37] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem, 2013. URL https://arxiv.org/abs/1201.0749.

[38] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

[39] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pp. 8162–8171. PMLR, 2021.

[40] Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Gunter Klambauer. Fréchet chemnet distance: a metric for generative models for molecules in drug discovery. *Journal of chemical information and modeling*, 58(9):1736–1741, 2018.

[41] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1): 140022, 2014.

[42] RDKit contributors. RDKit: Open-source cheminformatics software, 2026. URL https://www.rdkit.org/. Evaluation version 2026.3.5.

[43] Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. In *International Conference on Learning Representations (ICLR)*, 2023.

[44] Steven Rosenberg. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. Number 31. Cambridge University Press, 1997.

[45] Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. Diffusionnet: Discretization agnostic learning on surfaces. *ACM Trans. Graph.*, 41(3), mar 2022.

[46] Tianchang Shen, Zhaoshuo Li, Marc Law, Matan Atzmon, Sanja Fidler, James Lucas, Jun Gao, and Nicholas Sharp. Spacemesh: A continuous representation for learning manifold surface meshes. In *SIGGRAPH Asia 2024 Conference Papers*, pp. 1–11, 2024.

[47] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*, volume 4. Citeseer, 1998.

[48] Anthony Stephenson, Ian Gallagher, and Christopher Nemeth. Generator-based graph generation via heat diffusion, 2026. URL https://arxiv.org/abs/2602.03612.

[49] Kiwon Um, Robert Brand, Yun Raymond Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in neural information processing systems*, 33:6111–6122, 2020.

[50] 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.

[51] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UaAD-Nu86WX.

[52] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

[53] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[54] Po-Wei Wang, Priya L. Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver, 2019. URL https://arxiv.org/abs/1905.12149.

[55] Yudong W. Xu, Wenhao Li, Scott Sanner, and Elias B. Khalil. Self-supervised transformers as iterative solution improvers for constraint satisfaction, 2025. URL https://arxiv.org/abs/2502.15794.

[56] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

A Connection between kernel functions and graph Laplacians

We briefly explain the connection between kernel functions [47], such as the RBF kernel, and graph Laplacians. For more details, we refer the reader to [5, 44]. For simplicity of exposition, we assume that our initial node representations are n points lying in a d-dimensional Euclidean space $\mathbb{R}^d$, but this can be extended to more general Riemannian manifolds [44].

Let $\mathbf{\Theta} \in \mathbb{R}^{n \times d}$ denote a set of n points in $\mathbb{R}^d$, and let $\boldsymbol{\theta}_i^\top \in \mathbb{R}^d$ denote the i-th row of $\mathbf{\Theta}$.

Let $f_0 : \mathbb{R}^d \to \mathbb{R}$ denote the initial temperature distribution on $\mathbb{R}^d$. For $t > 0$, we define $f : (0, +\infty) \times \mathbb{R}^d \to \mathbb{R}$ by

$$f(t, \boldsymbol{\theta}) := \int_{\mathbb{R}^d} \frac{1}{(4\pi t)^{d/2}} e^{-\frac{1}{4t}\|\boldsymbol{\theta} - \mathbf{y}\|_2^2} f_0(\mathbf{y}) \mathrm{d}\mathbf{y}. \tag{11}$$

Under suitable assumptions, such as $f_0 \in L^1(\mathbb{R}^d)$, $f(t, \cdot)$ converges to f_0 in $L^1(\mathbb{R}^d)$ as $t \to 0^+$; convergence also holds pointwise at every Lebesgue point of f_0 [44]. We have $\forall \boldsymbol{\theta} \in \mathbb{R}^d$, $f(t, \boldsymbol{\theta}) \to 0$ as $t \to +\infty$. The corresponding integral kernel for the heat equation on $\mathbb{R}^d$ (called *heat kernel*) is:

$$k(t, \boldsymbol{\theta}, \mathbf{y}) := \frac{1}{(4\pi t)^{d/2}} e^{-\frac{1}{4t}\|\boldsymbol{\theta} - \mathbf{y}\|_2^2} \tag{12}$$

It is a smooth function on $(0, +\infty) \times \mathbb{R}^d \times \mathbb{R}^d$ and is proportional to a standard RBF kernel when t is fixed.

For a bandwidth $h \in (0, +\infty)$, let us construct a heat-kernel matrix $\mathbf{K}^h \in \mathbb{R}^{n \times n}$ such that $\forall i,\ \mathbf{K}^h_{i,i} = 0$, and $\forall i \neq j,\ \mathbf{K}^h_{i,j} = \eta_h k(h, \boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$ where $\eta_h := (4\pi h)^{d/2}$.

A.1 Standard Laplacian

We can construct the corresponding (negative of the) standard Laplacian matrix:

$$\mathbf{L}^h_{\gamma,\delta} := \gamma \left(\frac{1}{n} \left(\mathbf{K}^h - \mathrm{diag}(\mathbf{K}^h \mathbf{1}) \right) - \delta \mathbf{I} \right) \tag{13}$$

$\mathbf{K}^h$ is symmetric, so all its eigenvalues are real. All its entries lie in the interval $[0, 1]$. Let $d_i := \sum_{j \neq i} \mathbf{K}^h_{ij} \leq n - 1$. Row i of $\mathbf{K}^h - \mathrm{diag}(\mathbf{K}^h \mathbf{1})$ has diagonal entry $-d_i$ and off-diagonal absolute row sum d_i. The Gershgorin circle theorem therefore allows us to bound its eigenvalues in $[-2(n-1), 0]$. Consequently, the eigenvalues of $\mathbf{L}^h_{\gamma,\delta}$ are in $[\gamma(\frac{-2(n-1)}{n} - \delta), -\gamma\delta] \subset [\gamma(-2 - \delta), -\gamma\delta]$ for all $h > 0$.

Since we have $\mathbf{L}^h_{\gamma,0} \mathbf{1} = \mathbf{0}$ for all $h > 0$, the unit eigenvector $\mathbf{v}_h$ of $\mathbf{L}^h_{\gamma,0}$ corresponding to its 0 eigenvalue can be chosen as $\frac{1}{\sqrt{n}}\mathbf{1}$ for all $h > 0$. Therefore, for every $h > 0$, we have:

$$\lim_{s \to +\infty} e^{s\mathbf{L}^h_{\gamma,0}} = \mathbf{v}_h \mathbf{v}_h^\top = \frac{1}{n} \mathbf{1}\mathbf{1}^\top \tag{14}$$

In other words, for any vector $\mathbf{y} = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$, $\lim_{s \to +\infty} e^{s\mathbf{L}^h_{\gamma,0}} \mathbf{y}$ is a vector whose elements are all equal to the average of $\mathbf{y}$ (i.e., $\frac{1}{n}\sum_{i=1}^n y_i$).

A.2 Normalised Laplacian

For simplicity of exposition, we consider that $\gamma = 1$. We can construct the (negative of the) symmetric normalised Laplacian matrix:

$$\mathbf{L}^h_{\mathrm{sym},\delta} := \mathbf{D}^{-1/2} \mathbf{K}^h \mathbf{D}^{-1/2} - (1 + \delta)\mathbf{I}, \text{ where } \mathbf{D} := \mathrm{diag}(\mathbf{K}^h \mathbf{1}) \tag{15}$$

As h tends to $+\infty$, all the non-diagonal elements of $\mathbf{K}^h$ tend to 1. The unit eigenvector $\mathbf{v}_h$ with positive entries corresponding to the 0 eigenvalue of $\mathbf{L}^h_{\mathrm{sym},0}$ tends to $\frac{1}{\sqrt{n}}\mathbf{1}$ as h tends to $+\infty$. Therefore, we have:

$$\lim_{h \to +\infty} \left(\lim_{s \to +\infty} e^{s\mathbf{L}^h_{\mathrm{sym},0}} \right) = \lim_{h \to +\infty} \mathbf{v}_h \mathbf{v}_h^\top = \frac{1}{n} \mathbf{1}\mathbf{1}^\top \tag{16}$$

In the case of the symmetric normalised Laplacian, $\mathbf{v}_h$ need not equal $\frac{1}{\sqrt{n}}\mathbf{1}$ for finite $h > 0$.

B Solving the forward SDE

As an illustration, we explain how to solve Eq. (1) directly when $\mathbf{a} = \mathbf{0}$ and $\mathbf{\Delta} = \mathbf{L}$. The more general case follows the same idea. Let us define $\mathbf{\Phi}_{t,t_0}$ as the fundamental matrix written as follows:

$$\mathbf{\Phi}_{t,t_0} = e^{(t-t_0)\mathbf{L}} \iff \mathbf{\Phi}_{t,t_0}^{-1} = e^{(t_0-t)\mathbf{L}} \tag{17}$$

The solution of Eq. (2) when $\mathbf{a} = \mathbf{0}$ is:

$$\mathbf{x}(t) = \mathbf{\Phi}_{t,t_0}\left(\mathbf{x}(t_0) + \mathbf{B}\int_{t_0}^{t} \mathbf{\Phi}_{s,t_0}^{-1}\mathrm{d}\mathbf{w}(s)\right) = \mathbf{\Phi}_{t,t_0}\mathbf{x}(t_0) + \mathbf{B}\mathbf{\Phi}_{t,t_0}\int_{t_0}^{t} \mathbf{\Phi}_{s,t_0}^{-1}\mathrm{d}\mathbf{w}(s) \tag{18}$$

The second term can be approximated by means of the following sums for $t_0 \leq t_1 \leq \cdots \leq t_T = t$:

$$\mathbf{x}(t) \approx \mathbf{\Phi}_{t,t_0}\mathbf{x}(t_0) + \mathbf{B}e^{(t-t_0)\mathbf{L}}\sum_{i=1}^{T} e^{(t_0-\xi_i)\mathbf{L}}\left(\mathbf{w}(t_i) - \mathbf{w}(t_{i-1})\right),\ t_{i-1} \leq \xi_i \leq t_i \tag{19}$$

We simplify the above equation by considering that $t_0 = 0$, $t_T = t$, and $\forall i, \xi_i = t_{i-1}$:

$$\mathbf{\Phi}_{t,t_0}\mathbf{x}(t_0) + \mathbf{B}e^{(t-t_0)\mathbf{L}}\sum_{i=1}^{T} e^{(t_0-\xi_i)\mathbf{L}}\left(\mathbf{w}(t_i) - \mathbf{w}(t_{i-1})\right) \tag{20}$$

$$= e^{t\mathbf{L}}\mathbf{x}(0) + \mathbf{B}e^{t\mathbf{L}}\sum_{i=1}^{T} e^{-t_{i-1}\mathbf{L}}\left(\mathbf{w}(t_i) - \mathbf{w}(t_{i-1})\right) \tag{21}$$

$$= e^{t\mathbf{L}}\mathbf{x}(0) + \sum_{i=1}^{T} \mathbf{B}e^{(t-t_{i-1})\mathbf{L}}\left(\mathbf{w}(t_i) - \mathbf{w}(t_{i-1})\right) \tag{22}$$

$$= e^{t\mathbf{L}}\mathbf{x}(0) + \mathbf{B}\left(e^{t\mathbf{L}}\mathbf{w}(t_1) + e^{(t-t_1)\mathbf{L}}(\mathbf{w}(t_2) - \mathbf{w}(t_1)) + \cdots + e^{(t-t_{T-1})\mathbf{L}}(\mathbf{w}(t_T) - \mathbf{w}(t_{T-1}))\right) \tag{23}$$

We illustrate in Section B.3 how to implement the above approximation in MATLAB. This can be very expensive to compute.

B.1 Mean vector

We have the following mean vector at time $s + t$:

$$\frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbf{L}\boldsymbol{\mu} + \mathbf{a} \text{ with initial condition } s \geq 0,\ \boldsymbol{\mu}_s(s, \mathbf{x}, \mathbf{L}) = \mathbf{x}(s). \tag{24}$$

Its solution is formulated as follows:

$$\forall s \geq 0, t \geq 0, \boldsymbol{\mu}_{s+t}(s, \mathbf{x}, \mathbf{L}) = e^{t\mathbf{L}}\mathbf{x}(s) + e^{t\mathbf{L}}\left(\int_0^t e^{-u\mathbf{L}}\mathrm{d}u\right)\mathbf{a} \tag{25}$$

$$= e^{t\mathbf{L}}\left(\mathbf{x}(s) + \left[-e^{-u\mathbf{L}}\mathbf{L}^{-1}\right]_0^t\mathbf{a}\right) \tag{26}$$

$$= e^{t\mathbf{L}}\left(\mathbf{x}(s) + (\mathbf{I} - e^{-t\mathbf{L}})\mathbf{L}^{-1}\mathbf{a}\right) = e^{t\mathbf{L}}\mathbf{x}(s) + (e^{t\mathbf{L}} - \mathbf{I})\mathbf{L}^{-1}\mathbf{a} \tag{27}$$

$$= e^{t\mathbf{L}}\left(\mathbf{x}(s) + \mathbf{L}^{-1}\mathbf{a}\right) - \mathbf{L}^{-1}\mathbf{a} \tag{28}$$

B.2 Covariance matrix

The covariance matrix is (see Section 9.2 (Example 9.2.12) of [4] for details):

$$\forall s \geq 0, \forall t \geq 0, \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \mathbf{B}(2\mathbf{L})^{-1}(e^{2t\mathbf{L}} - \mathbf{I})\mathbf{B} = \frac{1}{2}\mathbf{B}\mathbf{L}^{-1}(e^{2t\mathbf{L}} - \mathbf{I})\mathbf{B} \tag{29}$$

Let $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ be the eigendecomposition of $\mathbf{L}$ where $\mathbf{\Lambda}$ is a diagonal matrix whose diagonal elements are all negative. We show that if $\mathbf{B} = (-2\mathbf{L})^{1/2}$, then $\frac{1}{2}\mathbf{B}\mathbf{L}^{-1}(e^{2t\mathbf{L}} - \mathbf{I})\mathbf{B} = \mathbf{I} - e^{2t\mathbf{L}}$.

The eigendecomposition of $\mathbf{B}$ is then $\mathbf{B} = \sqrt{2}\mathbf{U}(-\mathbf{\Lambda})^{1/2}\mathbf{U}^\top$, $e^{2t\mathbf{L}} = \mathbf{U}e^{2t\mathbf{\Lambda}}\mathbf{U}^\top$ and $\mathbf{I} = \mathbf{U}\mathbf{U}^\top = \mathbf{U}^\top\mathbf{U}$. We also recall that the multiplication of two diagonal matrices is always commutative.

$$\frac{1}{2}\mathbf{B}\mathbf{L}^{-1}(e^{2t\mathbf{L}} - \mathbf{I})\mathbf{B} = \frac{\sqrt{2}\sqrt{2}}{2}\mathbf{U}(-\mathbf{\Lambda})^{1/2}\mathbf{U}^\top\mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^\top\mathbf{U}(e^{2t\mathbf{\Lambda}} - \mathbf{I})\mathbf{U}^\top\mathbf{U}(-\mathbf{\Lambda})^{1/2}\mathbf{U}^\top \tag{30}$$

$$= -\mathbf{U}(-\mathbf{\Lambda})^{-1/2}(e^{2t\mathbf{\Lambda}} - \mathbf{I})(-\mathbf{\Lambda})^{1/2}\mathbf{U}^\top \tag{31}$$

$$= -\mathbf{U}(-\mathbf{\Lambda})^{-1/2}(-\mathbf{\Lambda})^{1/2}(e^{2t\mathbf{\Lambda}} - \mathbf{I})\mathbf{U}^\top \quad \text{(commutativity of diagonal matrices)} \tag{32}$$

$$= -\mathbf{U}(e^{2t\mathbf{\Lambda}} - \mathbf{I})\mathbf{U}^\top = \mathbf{I} - e^{2t\mathbf{L}}. \tag{33}$$

If $\mathbf{L}$ is symmetric negative definite, then we have $\lim_{t\to+\infty} e^{2t\mathbf{L}} = \mathbf{0}$. We then have:

$$\mathbf{B} = (-2\mathbf{L})^{1/2} \implies \forall s \geq 0, \lim_{t\to+\infty} \mathbf{\Sigma}_{s+t}(s, \mathbf{x}, \mathbf{L}) = \lim_{t\to+\infty} \left(\mathbf{I} - e^{2t\mathbf{L}}\right) = \mathbf{I}. \tag{34}$$

B.3 MATLAB code

We implement the second term of Eq. (23) in MATLAB. We recall that the first term is equal to $\boldsymbol{\mu}_t(0, \mathbf{x}, \mathbf{L})$ when $\mathbf{a} = \mathbf{0}$. Therefore, the expected value of the second term is $\mathbf{0}$, and the covariance matrix of the exact stochastic integral is $\mathbf{\Sigma}_t(0, \mathbf{x}, \mathbf{L})$. For a finite discretisation, the second term in Eq. (23) is an approximation whose covariance converges to $\mathbf{\Sigma}_t(0, \mathbf{x}, \mathbf{L})$ as the discretisation is refined. We verify this experimentally.

```
n = 3;
gamma = 0.2;
R = rand(n,n);
B = gamma * eye(n,n) + R * R';
L = -0.5 * B * B;
nb_iterations = 10000;

steps = 10000;
tau = 0.000001;
tn = tau * (steps+1);
target_covariance = eye(n,n) - expm(2 * L * tn);

list_of_results = [];
t = 0:tau:(tau*steps);

T = tn - t;
V = [];
for i=1:length(T)
    V = [V, expm(T(i) * L)];
end

for iter=1:nb_iterations
    randnmatrix = cumsum([zeros(n,1), sqrt(tau) * randn(n,steps
        +1)],2);
    w = randnmatrix(:,1:(end-1));
    w2 = randnmatrix(:,2:(end));
    W = w2 - w;
    result = B * V * reshape(W, [size(W,1)*size(W,2),1]);
    list_of_results = [list_of_results; result'];
end

abs(mean(list_of_results))
cov(list_of_results)
target_covariance
```

C Reverse-time process

C.1 Reverse-Time SDE

Rewriting Equation 1 in terms of $\mathbf{L}$ and with $\mathbf{a} = \mathbf{0}$, we get

$$\mathrm{d}\mathbf{x}(t) = \mathbf{L}\mathbf{x}(t)\mathrm{d}t + (-2\mathbf{L})^{1/2}\mathrm{d}\mathbf{w}(t). \tag{35}$$

Recall that the starting condition $\mathbf{x}(0)$ and $\mathbf{L}$ are drawn together. We will now analyse the dynamics for a fixed $\mathbf{L}$. Let $p_t(\mathbf{x}(t)|\mathbf{L})$ be the corresponding conditional distribution of $\mathbf{x}(t)$ implied by Equation 35, when the initial condition, which is sampled conditional on $\mathbf{L}$, is marginalised out. This SDE can be reversed in time. More specifically, if we define $\mathbf{y}(t) = \mathbf{x}(t_T - t)$, then this process satisfies the SDE

$$\mathrm{d}\mathbf{y}(t) = -\mathbf{L}\left(\mathbf{y}(t) + 2\nabla \log p_{t_T - t}(\mathbf{y}(t)|\mathbf{L})\right)\mathrm{d}t + (-2\mathbf{L})^{1/2}\mathrm{d}\bar{\mathbf{w}}(t), \tag{36}$$

where the backward SDE is initialised at $\mathbf{y}(0) = \mathbf{x}(t_T) \sim p_{t_T}(\cdot \mid \mathbf{L})$ and $\bar{\mathbf{w}}$ is another Wiener process. Note that this is the standard time-reversal used in diffusion models [3, 18], except we have now conditioned the dynamics on $\mathbf{L}$. Diffusion models aim to learn the score function, which in our case is given by $\nabla \log p_t(\,\cdot\,|\mathbf{L})$. Thus, a naïve attempt to adapt standard diffusion models to our setting requires $\mathbf{L}$ both as an input to the score network and to evaluate the backward dynamics. This is difficult in our proposed setting since we do not have access to $\mathbf{L}$ during inference; we will see in Appendix C.3 how using flow matching addresses the issue.

C.2 Closed-Form Conditionals

When given access to $\mathbf{L}$ during inference, as is the case for Hoogeboom & Salimans [20], standard score matching can be applied through the continuous SDE formulation from Appendix C.1. Equivalently, discrete-time diffusion models [19] can be used as well. In what follows, we show that all the standard conditionals needed to train standard diffusion models are also available for our model, provided we also condition on $\mathbf{L}$. The derivations below are a rewriting of those carried out by Hoogeboom & Salimans [20] and enable training a model for the case where $\mathbf{L}$ is known during inference.

The conditional distribution given a clean sample $(\mathbf{x}(t_0), \mathbf{L})$ and some noisy representation $\mathbf{x}(t_{i+1})$ is a normal distribution $\mathcal{N}(\boldsymbol{\mu}_{q,i}, \boldsymbol{\Sigma}_{q,i}) := q(\mathbf{x}(t_i)|\mathbf{x}(t_{i+1}), \mathbf{x}(t_0), \mathbf{L})$ where $\boldsymbol{\mu}_{q,i}$ and $\boldsymbol{\Sigma}_{q,i}$ can be written in closed form (see Eq. (42) and Eq. (43)). These clean-sample-conditioned posteriors are available as training-time targets because $\mathbf{x}(t_0)$ is then known. Sequential sampling from these exact posteriors would require already knowing $\mathbf{x}(t_0)$ and therefore does not define a generative inference procedure. At inference time, a learned denoiser must instead approximate $q(\mathbf{x}(t_i)|\mathbf{x}(t_{i+1}), \mathbf{L})$, in which the unknown clean sample has been marginalised out. Such a denoiser would also require $\mathbf{L}$ as input and in the reverse dynamics, but we assume that $\mathbf{L}$ is not available at inference time.

We now derive the parameters $(\boldsymbol{\mu}_{q,i}, \boldsymbol{\Sigma}_{q,i})$.

To simplify the notation, we consider in the following that $t_0 = 0$ and $\mathbf{a} = \mathbf{0}$. We also define $\chi_i := t_{i+1} - t_i > 0$.

We also have:

$$\mathcal{N}(\boldsymbol{\mu}_{q,i}, \boldsymbol{\Sigma}_{q,i}) := q(\mathbf{x}(t_i)|\mathbf{x}(t_{i+1}), \mathbf{x}(t_0), \mathbf{L}) = \frac{q(\mathbf{x}(t_{i+1})|\mathbf{x}(t_i), \mathbf{x}(t_0), \mathbf{L})q(\mathbf{x}(t_i)|\mathbf{x}(t_0), \mathbf{L})}{q(\mathbf{x}(t_{i+1})|\mathbf{x}(t_0), \mathbf{L})} \tag{37}$$

$$\propto q(\mathbf{x}(t_{i+1})|\mathbf{x}(t_i), \mathbf{L})q(\mathbf{x}(t_i)|\mathbf{x}(t_0), \mathbf{L}) \tag{38}$$

Eq. (37) follows from Bayes' rule, and Eq. (38) follows because the denominator is constant with respect to $\mathbf{x}(t_i)$ and our forward process is Markov conditional on $\mathbf{L}$, so we have:

$$q(\mathbf{x}(t_{i+1})|\mathbf{x}(t_i), \mathbf{x}(t_0), \mathbf{L}) = q(\mathbf{x}(t_{i+1})|\mathbf{x}(t_i), \mathbf{L}) = \mathcal{N}(e^{\chi_i \mathbf{L}}\mathbf{x}(t_i), \mathbf{I} - e^{2\chi_i \mathbf{L}}) \tag{39}$$

We can exploit a Gaussian change-of-variable identity to rewrite the above equation, noting that the mean of $\mathbf{x}(t_{i+1})$ is a linear function of the form $e^{\chi_i \mathbf{L}}\mathbf{x}(t_i)$. Let us write:

$$\mathrm{Norm}_{\mathbf{z}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) := \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \boldsymbol{\mu})\right) \tag{40}$$

If the mean of a multivariate normal in $\mathbf{z}$ is a linear function $\mathbf{A}\mathbf{y} + \mathbf{b}$ of a second variable $\mathbf{y}$, we have:

$$\text{Norm}_{\mathbf{z}}(\mathbf{A}\mathbf{y} + \mathbf{b}, \boldsymbol{\Sigma}) \propto \text{Norm}_{\mathbf{y}}((\mathbf{A}^\top \boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}\mathbf{A}^\top \boldsymbol{\Sigma}^{-1}(\mathbf{z} - \mathbf{b}), (\mathbf{A}^\top \boldsymbol{\Sigma}^{-1}\mathbf{A})^{-1}) \tag{41}$$

One can combine Eq. (39) and Eq. (41) by considering $\boldsymbol{\Sigma} = \mathbf{I} - e^{2\chi_i \mathbf{L}}$, $\mathbf{A} = e^{\chi_i \mathbf{L}}$, $\mathbf{b} = \mathbf{0}$, $\mathbf{y} = \mathbf{x}(t_i)$ and $\mathbf{z} = \mathbf{x}(t_{i+1})$. This implies $\mathbf{A}^\top \boldsymbol{\Sigma}^{-1}\mathbf{A} = e^{2\chi_i \mathbf{L}}(\mathbf{I} - e^{2\chi_i \mathbf{L}})^{-1}$.

Since $q(\mathbf{x}(t_i)|\mathbf{x}(t_0), \mathbf{L}) = \mathcal{N}(e^{t_i \mathbf{L}}\mathbf{x}(t_0), \mathbf{I} - e^{2t_i \mathbf{L}})$, each of the two factors in Eq. (38) is then proportional to a normal distribution. We can follow standard formulae for the product of the probability density functions of two normal distributions. The product of any two normal densities $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$ is proportional to a normal density $\mathcal{N}(\boldsymbol{\mu}_3, \boldsymbol{\Sigma}_3)$ that satisfies the equalities: $\boldsymbol{\Sigma}_3^{-1} = \boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1}$, and $\boldsymbol{\mu}_3 = \boldsymbol{\Sigma}_3(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2)$.

The (inverse of the) covariance matrix $\boldsymbol{\Sigma}_{q,i}$ is formulated as follows:

$$\forall \chi_i > 0, t_i > 0,\ \boldsymbol{\Sigma}_{q,i}^{-1} = e^{2\chi_i \mathbf{L}}(\mathbf{I} - e^{2\chi_i \mathbf{L}})^{-1} + (\mathbf{I} - e^{2t_i \mathbf{L}})^{-1} \tag{42}$$

The mean vector $\boldsymbol{\mu}_{q,i}$ is written:

$$\boldsymbol{\mu}_{q,i} = \boldsymbol{\Sigma}_{q,i}\left(e^{\chi_i \mathbf{L}}(\mathbf{I} - e^{2\chi_i \mathbf{L}})^{-1}\mathbf{x}(t_{i+1}) + (\mathbf{I} - e^{2t_i \mathbf{L}})^{-1}e^{t_i \mathbf{L}}\mathbf{x}(t_0)\right) \tag{43}$$

C.3 Flow matching

This section explains how our approach can be adapted to flow matching [33].

C.3.1 Stochastic heat flow matching

Here we use standard flow matching notation and write $\bar{\mathbf{x}}_1 = \mathbf{x}(0)$ for a clean sample from the target distribution. We denote its corresponding negative Laplacian matrix by $\mathbf{L}$. Consider the transition kernel that is conditioned on $(\bar{\mathbf{x}}_1, \mathbf{L})$:

$$q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L}) := \mathcal{N}(\bar{\mathbf{x}}; \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1), \bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)), \tag{44}$$

where $\bar{\boldsymbol{\mu}}_{t,\mathbf{L}} : \mathbb{R}^n \to \mathbb{R}^n$ and $\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}} : \mathbb{R}^n \to \mathcal{S}_+^n$ are known functions, and where $\mathcal{S}_+^n$ denotes the space of $n \times n$ symmetric positive definite matrices. Throughout this derivation, we restrict $t \in [0, 1)$ so that $\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)$ remains positive definite. Note that this transition kernel is similar to usual transition kernels arising in score-based diffusion models and flow matching, except that its covariance matrix is anisotropic and depends on $\bar{\mathbf{x}}_1$ indirectly through the sample-specific matrix $\mathbf{L}$ associated with $\bar{\mathbf{x}}_1$. The transition kernel arising from the SDE in Eq. (1) has precisely this form. We will also denote the noisy marginals as q_t, i.e.

$$q_t(\bar{\mathbf{x}}) = \int\int q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})q_1(\bar{\mathbf{x}}_1, \mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L}, \tag{45}$$

where we slightly abuse notation and, depending on context, q_1 can refer to the data distribution (when used as $q_1(\bar{\mathbf{x}}_1)$) or to the joint of data and $\mathbf{L}$ (when used as $q_1(\bar{\mathbf{x}}_1, \mathbf{L})$). For any fixed $\bar{\mathbf{x}}_1$, we want to find a vector field $\mathbf{u}_t(\cdot|\bar{\mathbf{x}}_1, \mathbf{L})$ generating this transition kernel, i.e., it should obey the continuity equation:

$$\frac{\partial q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})}{\partial t} + \text{Tr}\left(\nabla_{\bar{\mathbf{x}}}[q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})]\right) = 0. \tag{46}$$

Note that we can rewrite the continuity equation as

$$\frac{\partial \log q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})}{\partial t} + \text{Tr}(\nabla_{\bar{\mathbf{x}}}\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})) + \mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L})^\top \nabla_{\bar{\mathbf{x}}} \log q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L}) = 0. \tag{47}$$

Since $\log q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)| - \frac{1}{2}(\bar{\mathbf{x}} - \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top \bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}} - \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))$, where $|\cdot|$ denotes the determinant of a matrix, we have that

$$\nabla_{\bar{\mathbf{x}}} \log q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1, \mathbf{L}) = -\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}} - \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)). \tag{48}$$

To simplify notation, we will write the time derivative of a function f as $f'(t) = \frac{\partial f(t)}{\partial t}$. Recall that, from matrix calculus, we have that $\frac{\partial}{\partial t}\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1) = -\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)$ and that

$\frac{\partial}{\partial t}\log|\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)| = \mathrm{Tr}(\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))$. It then follows that

$$\frac{\partial \log q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})}{\partial t} = -\frac{1}{2}\,\mathrm{Tr}\left(\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\right) - \frac{1}{2}\frac{\partial}{\partial t}\left[(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))\right] \tag{49}$$

$$= -\frac{1}{2}\,\mathrm{Tr}\left(\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\right) - \frac{1}{2}\Big[-2(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) \tag{50}$$

$$-(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))\Big] \tag{51}$$

$$= -\frac{1}{2}\,\mathrm{Tr}\left(\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\right) + (\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) \tag{52}$$

$$+\frac{1}{2}(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)). \tag{53}$$

Plugging everything back into Eq. (47), we get that $\mathbf{u}_t(\cdot|\bar{\mathbf{x}}_1,\mathbf{L})$ must satisfy:

$$\begin{aligned}
&-\frac{1}{2}\,\mathrm{Tr}\left(\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\right) + (\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\\
&+\frac{1}{2}(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1))\\
&+\mathrm{Tr}(\nabla_{\bar{\mathbf{x}}}\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})) - \mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})^\top\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)) = 0.
\end{aligned} \tag{54}$$

Now, we define

$$\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L}) \coloneqq \bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) + \frac{1}{2}\bar{\boldsymbol{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\boldsymbol{\Sigma}}_{t,\mathbf{L}}^{-1}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}-\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)), \tag{55}$$

which satisfies Eq. (54). From a standard flow matching argument, it follows that training a vector field net_θ to minimise

$$\mathbb{E}_{t,(\bar{\mathbf{x}}_1,\mathbf{L})\sim q_1,\bar{\mathbf{x}}_t\sim q_{t|1}(\cdot|\bar{\mathbf{x}}_1,\mathbf{L})}\left[\|\,\mathrm{net}_\theta(\bar{\mathbf{x}}_t,t) - \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1,\mathbf{L})\|_2^2\right] \tag{56}$$

and then solving the ODE

$$\mathrm{d}\bar{\mathbf{x}}_t = \mathrm{net}_{\theta^*}(\bar{\mathbf{x}}_t,t)\mathrm{d}t \tag{57}$$

initialised at $\bar{\mathbf{x}}_0 \sim q_0$ will result in $\bar{\mathbf{x}}_1 \sim q_1$ (assuming net_{θ^*} has achieved its population-level optimum in Eq. (56)). To see that this is indeed the case, we take the expectation of Eq. (46) *w.r.t.* q_1:

$$\int\int\left(\frac{\partial q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})}{\partial t} + \mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}[q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})]\right)\right)q_1(\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L} = 0. \tag{58}$$

First, note that

$$\int\int\frac{\partial q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})}{\partial t}q_1(\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L} = \frac{\partial}{\partial t}\int\int q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})q_1(\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L} = \frac{\partial}{\partial t}q_t(\bar{\mathbf{x}}). \tag{59}$$

Then, note that

$$\int\int\mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}[q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})]\right)q_1(\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L} \tag{60}$$

$$= \mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}\int\int q_{t|1}(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})q_1(\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L}\right) \tag{61}$$

$$= \mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}\int\int q_{1|t}(\bar{\mathbf{x}}_1,\mathbf{L}|\bar{\mathbf{x}})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})q_t(\bar{\mathbf{x}})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L}\right) \tag{62}$$

$$= \mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}\left[q_t(\bar{\mathbf{x}})\int\int q_{1|t}(\bar{\mathbf{x}}_1,\mathbf{L}|\bar{\mathbf{x}})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L}\right]\right) \tag{63}$$

$$= \mathrm{Tr}\left(\nabla_{\bar{\mathbf{x}}}\left[q_t(\bar{\mathbf{x}})\mathbf{u}_t(\bar{\mathbf{x}})\right]\right), \tag{64}$$

where $q_{1|t}(\bar{\mathbf{x}}_1,\mathbf{L}|\bar{\mathbf{x}})$ denotes the joint posterior of $(\bar{\mathbf{x}}_1,\mathbf{L})$ given $\bar{\mathbf{x}}_t = \bar{\mathbf{x}}$, and where we defined $\mathbf{u}_t(\bar{\mathbf{x}}) \coloneqq \int\int q_{1|t}(\bar{\mathbf{x}}_1,\mathbf{L}|\bar{\mathbf{x}})\mathbf{u}_t(\bar{\mathbf{x}}|\bar{\mathbf{x}}_1,\mathbf{L})\mathrm{d}\bar{\mathbf{x}}_1\mathrm{d}\mathbf{L}$. Combining these two observations, we get that the continuity equation

$$\frac{\partial}{\partial t}q_t(\bar{\mathbf{x}}) + \mathrm{Tr}(\nabla_{\bar{\mathbf{x}}}[q_t(\bar{\mathbf{x}})\mathbf{u}_t(\bar{\mathbf{x}})]) = 0 \tag{65}$$

Algorithm 3 Training of the flow matching algorithm

Require: hyperparameter $\delta > 0$ to make $\mathbf{L}$ symmetric negative definite
  while Not converged **do**
    Sample training graph containing $(\mathbf{x}_0 = \bar{\mathbf{x}}_1, \mathbf{L})$
    Sample $t \sim \text{Uniform}([0, 1))$, define $s := 1 - t \in (0, 1]$
    Draw $\bar{\mathbf{x}}_t \sim \mathcal{N}\left(e^{s\mathbf{L}}\bar{\mathbf{x}}_1, \mathbf{I} - e^{2s\mathbf{L}}\right)$
    $\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) = \mathbf{L}e^{2s\mathbf{L}}(\mathbf{I} - e^{2s\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{s\mathbf{L}}(\mathbf{I} - e^{2s\mathbf{L}})^{-1}\bar{\mathbf{x}}_1$
    minimise $\|\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) - \text{net}_\theta(\bar{\mathbf{x}}_t, t)\|_2^2$
  end while
  Return net_θ

holds. In turn, solving the ODE

$$\mathrm{d}\bar{\mathbf{x}}_t = \mathbf{u}_t(\bar{\mathbf{x}}_t)\mathrm{d}t \tag{66}$$

starting from $\bar{\mathbf{x}}_0 \sim q_0$ results in $\bar{\mathbf{x}}_1 \sim q_1$. Thus, the last thing we have to check is that the population-level optimiser of Eq. (56) is achieved when $\text{net}_{\theta^*}(\bar{\mathbf{x}}, t) = \mathbf{u}_t(\bar{\mathbf{x}})$. It is well known that MSE is minimised by the conditional expectation, so that in this case, the population-level optimiser of Eq. (56) is $\text{net}_{\theta^*}(\bar{\mathbf{x}}_t, t) = \mathbb{E}_{(\bar{\mathbf{x}}_1, \mathbf{L}) \sim q_{1|t}(\cdot|\bar{\mathbf{x}}_t)}[\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})]$, which is exactly how $\mathbf{u}_t(\bar{\mathbf{x}}_t)$ is defined. This ensures the correctness of the objective and sampling method.

Discussion. In general, reversing an SDE with transition kernel as in Eq. (44) would require the score function to have access to the covariance matrices that were used during the forward process (see Appendix C.1); this of course is not typically available during inference since we do not know $\bar{\mathbf{x}}_1$, nor $\mathbf{L}$, in advance to compute $\bar{\mathbf{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)$. What is interesting about this flow matching formulation is that we can still construct a vector field net_θ to properly sample from q_1 even though net_θ takes only $\bar{\mathbf{x}}_t$ and t as inputs, i.e., importantly, it does not require access to $\bar{\mathbf{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)$, nor to $\mathbf{L}$. Note also that the ODE in Eq. (66) does not induce the same distribution over paths as the starting SDE which gives rise to the transition kernel, but that is not a problem: it does produce the same marginals q_t, and that is all we need for a correct method to sample from q_1. As with standard diffusion models or flow matching, we need q_0 to either be known exactly or be approximated. When we replace the exact finite-time q_0 by a standard normal distribution, the associated stochastic-heat conditional marginals are finite-time approximations.

C.3.2 Diffusion conditional velocity field

Let us define the time-dependent mean vector and covariance matrix as follows for all $t \in [0, 1]$:

$$\bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) := e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1, \qquad \bar{\mathbf{\Sigma}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) := \mathbf{I} - e^{2(1-t)\mathbf{L}} \tag{67}$$

With proper scaling of $\mathbf{L}_{\gamma,\delta} = \mathbf{L}$ (i.e., if $\gamma > 0$ is large enough), we have for $t = 0$:

$$\bar{\boldsymbol{\mu}}_{0,\mathbf{L}}(\bar{\mathbf{x}}_1) = e^{\mathbf{L}}\bar{\mathbf{x}}_1 \approx \mathbf{0}, \qquad \bar{\mathbf{\Sigma}}_{0,\mathbf{L}}(\bar{\mathbf{x}}_1) \approx \mathbf{I} \tag{68}$$

When $t = 1$, we have:

$$\bar{\boldsymbol{\mu}}_{1,\mathbf{L}}(\bar{\mathbf{x}}_1) = \bar{\mathbf{x}}_1, \qquad \bar{\mathbf{\Sigma}}_{1,\mathbf{L}}(\bar{\mathbf{x}}_1) = \mathbf{0} \tag{69}$$

We have the time derivatives of the mean vector and covariance matrix:

$$\bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) = -\mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1, \qquad \bar{\mathbf{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) = 2\mathbf{L}e^{2(1-t)\mathbf{L}} \tag{70}$$

We find the velocity field as formulated in Eq. (6):

$$\mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) = \frac{1}{2}\bar{\mathbf{\Sigma}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)\bar{\mathbf{\Sigma}}^{-1}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)(\bar{\mathbf{x}}_t - \bar{\boldsymbol{\mu}}_{t,\mathbf{L}}(\bar{\mathbf{x}}_1)) + \bar{\boldsymbol{\mu}}'_{t,\mathbf{L}}(\bar{\mathbf{x}}_1) \tag{71}$$

$$= \frac{1}{2}(2\mathbf{L}e^{2(1-t)\mathbf{L}})(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}(\bar{\mathbf{x}}_t - e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1) - \mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 \tag{72}$$

$$= \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1 - \mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 \tag{73}$$

$$= \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1 \tag{74}$$

This results in training our neural network using Algorithm 3, in which our neural network $\text{net}_\theta(\bar{\mathbf{x}}_t, t)$ tries to predict the velocity at time $t \in [0, 1)$ when given the current position $\bar{\mathbf{x}}_t$. The goal at inference time is to integrate from $t = 0$ to $t = 1$ by following the velocity over time.

C.4 Conditional average velocity

As explained in Section C.3, we have the following equality in the conditional case:

$$\bar{\mathbf{x}}'_t = \frac{\mathrm{d}\bar{\mathbf{x}}_t}{\mathrm{d}t} = \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L}) \tag{75}$$

where $\bar{\mathbf{x}}'_t = \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})$ is the conditional velocity field at time $t \in [0, 1)$ as defined in Eq. (6).

C.4.1 Solution

Our goal is to find a formulation of $\bar{\mathbf{x}}$ that recovers the values at the boundaries $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$ and $\bar{\mathbf{x}}(1) = \bar{\mathbf{x}}_1$ and satisfies Eq. (6) that is written as follows:

$$\forall t \in [0, 1),\ \bar{\mathbf{x}}'_t = \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1 \tag{76}$$

The following formulation satisfies the above conditions:

$$\forall t \in [0, 1],\ \bar{\mathbf{x}}(t) = \bar{\mathbf{x}}_t = e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 + (\mathbf{I} - e^{2(1-t)\mathbf{L}})^{1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1) \tag{77}$$

Proof: Let us first rewrite Eq. (77) to obtain the value of $\bar{\mathbf{x}}_0$ as follows:

$$\forall t \in [0, 1),\ \bar{\mathbf{x}}_0 = (\mathbf{I} - e^{2\mathbf{L}})^{1/2}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_t - e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1) + e^{\mathbf{L}}\bar{\mathbf{x}}_1 \tag{78}$$

If we differentiate Eq. (77) w.r.t. t and then substitute $\bar{\mathbf{x}}_0$ from Eq. (78), we obtain:

$$\begin{aligned}
\bar{\mathbf{x}}'_t &= -\mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 + \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1) && (79)\\
&= -\mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 + \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}(\bar{\mathbf{x}}_t - e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1) && (80)\\
&= \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 - \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 && (81)\\
&= \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}(\mathbf{I} - e^{2(1-t)\mathbf{L}} + e^{2(1-t)\mathbf{L}})(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1 && (82)\\
\bar{\mathbf{x}}'_t &= \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_t - \mathbf{L}e^{(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1 && (83)
\end{aligned}$$

Eq. (77) also satisfies $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$ and $\bar{\mathbf{x}}(1) = \bar{\mathbf{x}}_1$, which completes the proof. □

C.4.2 Uniqueness of the solution

Eq. (76) can be written as the following linear first-order ODE with initial condition $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$:

$$\frac{\mathrm{d}\bar{\mathbf{x}}}{\mathrm{d}t} - \mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}} = -\mathbf{L}e^{(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1}\bar{\mathbf{x}}_1 \ \text{ s.t. } \bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0 \tag{84}$$

which guarantees a unique solution on $[0, 1)$ due to its continuity, and Eq. (77) satisfies $\bar{\mathbf{x}}(1) = \bar{\mathbf{x}}_1$.

C.4.3 Initial condition when t is positive

In some cases, the initial condition might be $\bar{\mathbf{x}}(a) = \bar{\mathbf{x}}_a$ for some $a \in (0, 1)$ instead of $\bar{\mathbf{x}}(0) = \bar{\mathbf{x}}_0$, and we might want to calculate $\bar{\mathbf{x}}_b$ for any $b \in [a, 1]$ given $(\bar{\mathbf{x}}_1, \mathbf{L})$.

Using Eq. (77) with $t = b$ and then substituting the expression from Eq. (78) with $t = a$, we can write $\bar{\mathbf{x}}_b$ as a function of $\bar{\mathbf{x}}_a$ and $\bar{\mathbf{x}}_1$ for all $a \in [0, 1)$ and all $b \in [a, 1]$:

$$\begin{aligned}
&\bar{\mathbf{x}}_a + \int_a^b \mathbf{u}_t(\bar{\mathbf{x}}_t|\bar{\mathbf{x}}_1, \mathbf{L})\mathrm{d}t = \bar{\mathbf{x}}_b = e^{(1-b)\mathbf{L}}\bar{\mathbf{x}}_1 + (\mathbf{I} - e^{2(1-b)\mathbf{L}})^{1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1)\\
=&e^{(1-b)\mathbf{L}}\bar{\mathbf{x}}_1 + (\mathbf{I} - e^{2(1-b)\mathbf{L}})^{1/2}(\mathbf{I} - e^{2(1-a)\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_a - e^{(1-a)\mathbf{L}}\bar{\mathbf{x}}_1) && (85)
\end{aligned}$$

C.4.4 Norm of the velocity field

We study the norm of the velocity field in Eq. (79) as a function of t.

$$\begin{aligned}
\|\bar{\mathbf{x}}'_t\|_2^2 =&\|\mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1\|_2^2 - 2\bar{\mathbf{x}}_1^\top\mathbf{L}^2e^{3(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1) && (86)\\
&+ \|\mathbf{L}e^{2(1-t)\mathbf{L}}(\mathbf{I} - e^{2(1-t)\mathbf{L}})^{-1/2}(\mathbf{I} - e^{2\mathbf{L}})^{-1/2}(\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1)\|_2^2
\end{aligned}$$

We consider in this section the large γ scaling under which $e^{\mathbf{L}}$ is nonzero but negligible ($e^{\mathbf{L}} \approx \mathbf{0}$). As $t \to 1^-$, the singular factors in Eq. (79) cause the velocity to diverge only when their residual is nonzero:

$$\bar{\mathbf{x}}_0 - e^{\mathbf{L}}\bar{\mathbf{x}}_1 \neq \mathbf{0} \quad \Longrightarrow \quad \lim_{t\to 1^-} \|\bar{\mathbf{x}}_t'\|_2^2 = +\infty. \tag{87}$$

This condition holds almost surely when $\bar{\mathbf{x}}_0$ is sampled from a Gaussian distribution (for fixed $\bar{\mathbf{x}}_1$ and $\mathbf{L}$), but it is not universal. If $\bar{\mathbf{x}}_0 = e^{\mathbf{L}}\bar{\mathbf{x}}_1$, the singular term vanishes, $\bar{\mathbf{x}}_t' = -\mathbf{L}e^{(1-t)\mathbf{L}}\bar{\mathbf{x}}_1$, and the endpoint limit is the finite value $\|\mathbf{L}\bar{\mathbf{x}}_1\|_2^2$.

There is likewise no general vanishing-velocity limit as $t \to 0^+$. For fixed $\mathbf{L}$ and inputs, $\lim_{t\to 0^+} \|\bar{\mathbf{x}}_t'\|_2^2 = \|\bar{\mathbf{x}}_0'\|_2^2$, whose magnitude depends on $\mathbf{L}$, $\bar{\mathbf{x}}_0$, and $\bar{\mathbf{x}}_1$. It is very small but generally nonzero under our chosen large γ scaling and for typical inputs with bounded magnitude, because the relevant matrix-exponential terms are strongly suppressed.

Fig. 8 illustrates the norm of $\bar{\mathbf{x}}_t'$ when $n = 5$, $\mathbf{L} = -10 \cdot \mathbf{I}$, $\bar{\mathbf{x}}_0$ is drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ as a finite-time approximation, and $\bar{\mathbf{x}}_1$ is sampled uniformly in $[-1, 1]^n$. The MATLAB code to generate the plot is also provided.

Our noise scheduler in Section 4.2 is motivated by the properties of the norm of the velocity field.

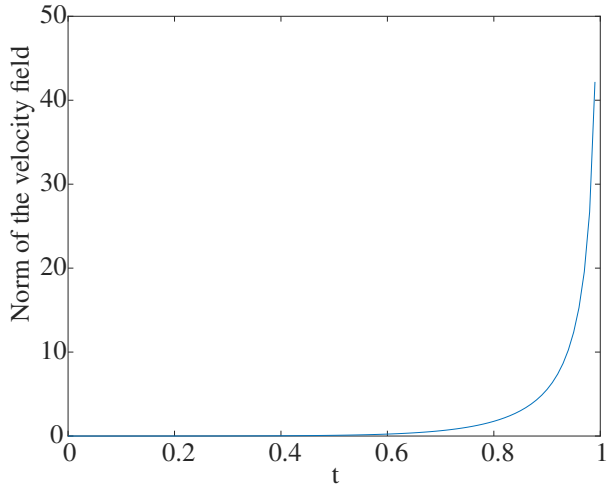


Figure 8: Norm of the velocity field $\|\bar{\mathbf{x}}_t'\|_2$ as a function of $t \in [0, 1)$ in the given toy example.

```
nb_iter = 1000; d = 5; step = 0.01; L = -10.0;
norm_of_gradient = 0.0

for i=1:nb_iter
    t = 0:step:(1-step);
    nb_steps = length(t);
    x0 = repmat(randn(d,1), 1, nb_steps);
    x1 = repmat(2*(rand(d,1) - 0.5), 1, nb_steps);
    q = sqrt(1-exp(2.*L));
    t = repmat(t, d, 1);
    a = (x0 - exp(L) .* x1);
    gt = - L .* exp((1-t).*L) .* x1 + L .* exp(2 * (1-t) .* L)
        .* a ./ (sqrt(1-exp(2*(1-t)*L)) * q);
    norm_of_gradient = norm_of_gradient + sqrt(sum(gt.^2,1));
end
plot(0:step:(1-step), norm_of_gradient / nb_iter)
xlabel("t")
ylabel("Norm of the velocity field")
```

D Graph generation

D.1 Methods

Notation. We denote by $\mathbf{v}_0 := \mathbf{1}$ an eigenvector of $\mathbf{L}_{\gamma,0}$ with eigenvalue equal to 0. We also define $\mathbf{V} := [\mathbf{v}_1, \dots, \mathbf{v}_k] \in \mathbb{R}^{n \times k}$ as a set of k eigenvectors of $\mathbf{L}_{\gamma,0}$ with eigenvalues of smallest magnitude excluding $\mathbf{v}_0$. Let $\boldsymbol{\lambda} \in \mathbb{R}^k$ contain the eigenvalues associated with the columns of $\mathbf{V}$. One can concatenate $\mathbf{V}$ to the node features $\mathbf{N}$ to form $\mathbf{X}(0) = [\mathbf{N}, \mathbf{V}]$ and reconstruct $\mathbf{V}$ in the same way as we reconstructed $\mathbf{N}$ previously. This is similar to the Laplacian positional encoding used for graph transformers [12, 13]. We present different alternatives to this approach.

Factorisation. One solution is to consider that the initial condition of our problem is $\bar{\mathbf{X}}_1 = \mathbf{X}(0) = [\mathbf{N}, \mathbf{Z}]$ where we define $\mathbf{Z} := \mathbf{V}\text{diag}(-\boldsymbol{\lambda})^{1/2}$. In this way, both the dominant eigenvectors and eigenvalues are taken into account during the generation process. We apply this approach to molecule generation on the QM9 dataset [41] in Appendix E.4.

Choice of eigenvectors. If $\mathbf{v}_i$ is an eigenvector, so is $-\mathbf{v}_i$. Thus, the 2^k matrices $[s_1\mathbf{v}_1, \dots, s_k\mathbf{v}_k]$, where each $s_i \in \{-1, 1\}$, represent equally valid sign choices for $\mathbf{V}$. One possible convention is to define the permutation-equivariant reference vector $\mathbf{d} := \mathbf{A}\mathbf{1} \geq \mathbf{0}$ and orient each selected eigenvector relative to it. Given any permutation matrix $\mathbf{P} \in \{0, 1\}^{n \times n}$ that leads to the node feature matrix $\mathbf{P}\mathbf{X}(0)$ and adjacency matrix $\mathbf{P}\mathbf{A}\mathbf{P}^\top$, we recover $\mathbf{P}\mathbf{A}\mathbf{P}^\top\mathbf{1} = \mathbf{P}\mathbf{A}\mathbf{1} = \mathbf{P}\mathbf{d}$ since $\mathbf{P}^\top\mathbf{1} = \mathbf{1}$. Whenever $\mathbf{v}_i^\top\mathbf{d} \neq 0$, choosing the i-th column of $\mathbf{V}$ as $\arg\max_{\mathbf{e} \in \{-\mathbf{v}_i, \mathbf{v}_i\}} \mathbf{e}^\top\mathbf{d}$ therefore gives a unique, permutation-equivariant orientation.

This convention is not guaranteed to be unique. In particular, for a regular graph, $\mathbf{d}$ is proportional to $\mathbf{1}$, while every nonconstant Laplacian eigenvector is orthogonal to $\mathbf{1}$, so $\mathbf{v}_i^\top\mathbf{d} = 0$. Repeated eigenvalues create a broader ambiguity because any orthogonal rotation within the corresponding eigenspace is valid; a sign rule alone cannot select a canonical basis. In our implementation, a numerically zero alignment is resolved by making the largest-magnitude entry positive, but this fallback does not eliminate ties or rotations within repeated eigenspaces. In order to reconstruct the whole graph, our denoiser neural network can contain one head that reconstructs nodes as explained in Section 4.2, and another head that reconstructs edges (whether binary, categorical, or with multi-dimensional feature vectors).

Sign-invariant neural network. Another solution is to use a sign-invariant and permutation-equivariant neural network such as SignNet [30] which is formulated as $\text{SignNet}(\mathbf{V}, \boldsymbol{\lambda}, \mathbf{N}) := \boldsymbol{\rho}\left([\boldsymbol{\phi}(\mathbf{v}_i, \lambda_i, \mathbf{N}) + \boldsymbol{\phi}(-\mathbf{v}_i, \lambda_i, \mathbf{N})]_{i=1}^k\right) \in \mathbb{R}^{n \times m}$ where $[]$ denotes concatenation, $\boldsymbol{\phi}$ and $\boldsymbol{\rho}$ are permutation-equivariant networks, and m is a hyperparameter that defines the output dimensionality of the model. The corresponding SignNet is invariant to the sign of the eigenvectors. We can formulate $\mathbf{X}(0) = [\mathbf{N}, \text{SignNet}(\mathbf{V}, \boldsymbol{\lambda}, \mathbf{N})]$ and train SignNet end-to-end in the same way as above.

Limitations and future work. The factor representation is not unique: $\mathbf{Z}\mathbf{Q}$ and $\mathbf{Z}$ have the same Gram matrix for any orthogonal matrix $\mathbf{Q}$, whereas coordinate-wise regression distinguishes between these factors. The sign convention above addresses only a subset of this ambiguity and does not resolve rotations within repeated eigenspaces. Future work could explore objectives that depend only on the invariant Gram matrix $\mathbf{Z}\mathbf{Z}^\top$, orthogonal Procrustes alignment between target and predicted factors, random right-orthogonal augmentation during training, or neural architectures that are invariant to both eigenvector signs and changes of basis within repeated eigenspaces.

D.2 Examples of generated graphs

Figs. 12 and 13 illustrate some graphs that were generated by reconstructing both the node features and some edge features derived from the eigendecomposition of the Laplacian matrix. In the case of 3-dimensional point clouds in Fig. 12, the graphs used as ground truth are on the left, and their adjacency matrices were returned by α-shapes [2, 14] to define the Laplacian matrix used for diffusion. The generated graphs are on the right. We see that our generated graphs share common patterns with the ground truth distribution. This approach could be generalised to form meshes as in [46] or graph metanetworks [31]. Fig. 13 illustrates some molecule graphs generated as explained in Appendix E.4.

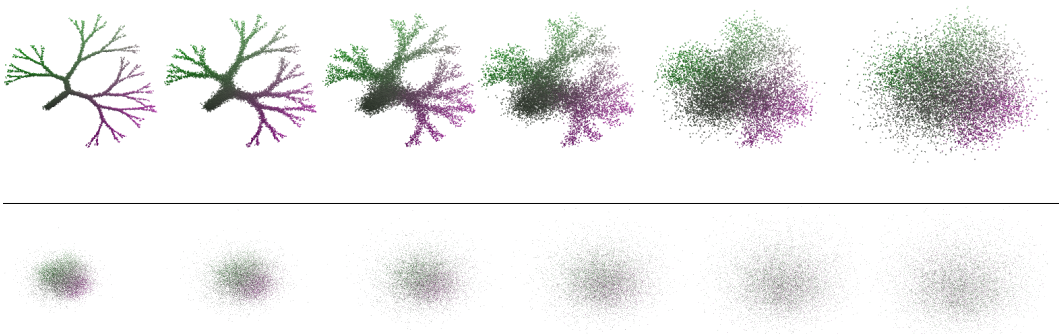


Figure 9: Clean sample $\bar{\mathbf{X}}_1 \in \mathbb{R}^{n\times 2}$ (top left) and noisy sample $\bar{\mathbf{X}}_0$ sampled from a standard normal distribution as a finite-time approximation (bottom right). The intermediate representations lie on the corresponding trajectory using our exponential scheduler with $\kappa = 5$. Each row represents a different zoom level of $\bar{\mathbf{X}}_t$.

E Experiments

We give additional details of the experiments.

E.1 Computer resources

Our model was tested on the following GPU models: NVIDIA RTX 6000 Ada and NVIDIA TITAN RTX. For Sudoku experiments with the best hyperparameter configuration, on an NVIDIA RTX 6000 Ada, $100{,}000$ training iterations take 2 h 40 min, which is $37{,}500$ iterations per hour; on an NVIDIA TITAN RTX, $100{,}000$ training iterations take 4 h 30 min, which is $22{,}222$ iterations per hour. These experiments do not require substantial CPU or RAM resources.

E.2 Toy experiment and point clouds

Fig. 10 and Fig. 11 illustrate the noising process under the linear and cosine noise schedulers defined in [19, 39]. Since Gaussian noise is injected independently across elements, the global structure of the data is not preserved as well as in our method. We use the neural network architecture described in Appendix E.3.

E.3 Sudoku

At inference time, the 27 row, column, and sub-grid channels are provided and held fixed as conditioning at every denoising step, so the model generates only the 9-dimensional digit-index part of $\mathbf{N} \in \mathbb{R}^{n\times d}$. If clues are given, their one-hot digit vectors are also provided and held fixed as conditioning at every step. After the final step, all elements of the generated digit matrix are clamped to the interval $[0, 1]$.

We also construct a Boolean matrix $\mathbf{R} \in \{0, 1\}^{81\times 9}$ whose 81 rows consist of nine copies of each of the nine one-hot digit vectors. Equivalently, $\mathbf{R}\mathbf{1} = \mathbf{1}$ and $\mathbf{R}^\top \mathbf{1} = 9\mathbf{1}$.

We calculate the distance matrix $\mathbf{S} \in \mathbb{R}^{81\times 81}$ that contains the pairwise squared Euclidean distances between the rows of $\mathbf{R}$ and the generated digit representations. The matrix $\mathbf{S}$ is used as the cost matrix of an assignment problem algorithm such as the Hungarian algorithm to assign each row of $\mathbf{R}$ (i.e., a digit instance) to a generated node. It is worth noting that this Sudoku postprocessing does not enforce Sudoku constraints; it only enforces that every digit occurs nine times globally. It does not enforce one occurrence per row, column, and sub-grid.

Neural network details. The trained denoiser consists of an input embedding layer, a series of Transformer encoder blocks, and a multi-layer perceptron (MLP). The linear embedding layer maps inputs into $\mathbb{R}^{d_{model}}$. The embedding is then fed into a three-layer Transformer with h attention heads and without positional encoding. The first feedforward layer in each Transformer block expands the dimensionality to $2 \times d_{model}$ with the ReLU activation function. The following MLP has N layers with d_{hidden} dimensions. We use a learning rate scheduler with $1{,}000$ iterations of linear warm-up

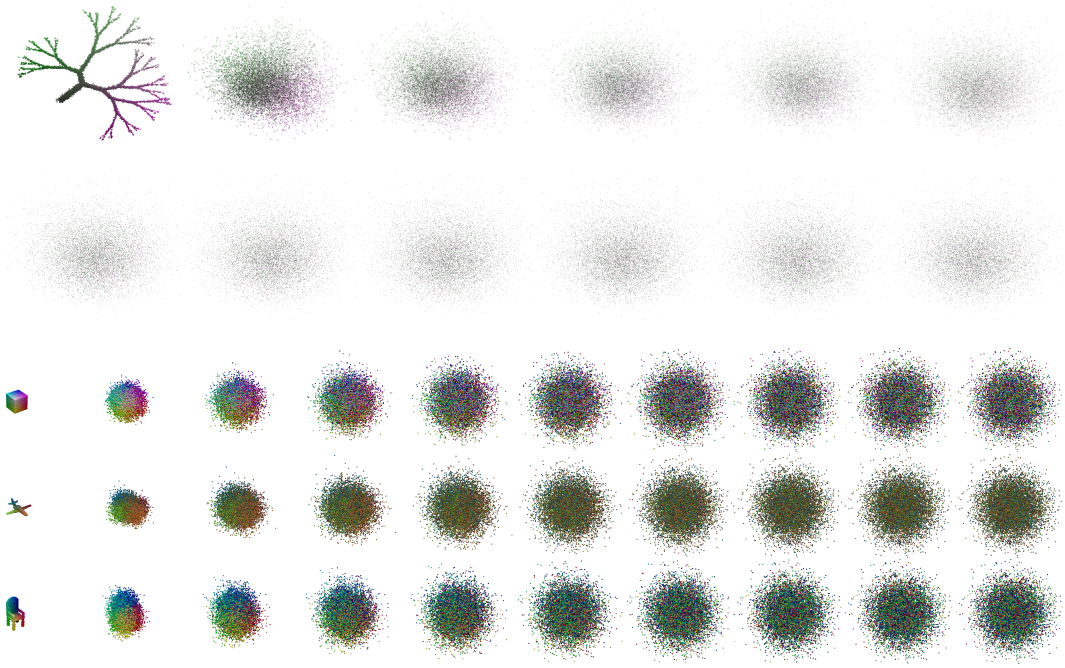


Figure 10: Linear scheduler as defined in [19]

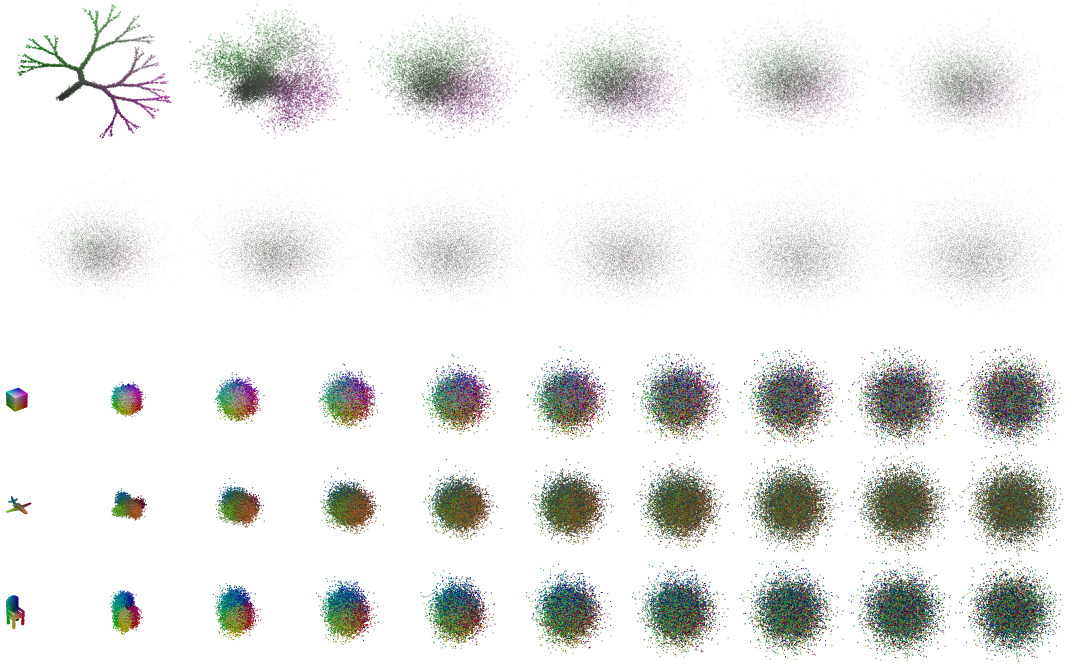


Figure 11: Cosine scheduler as defined in [39]

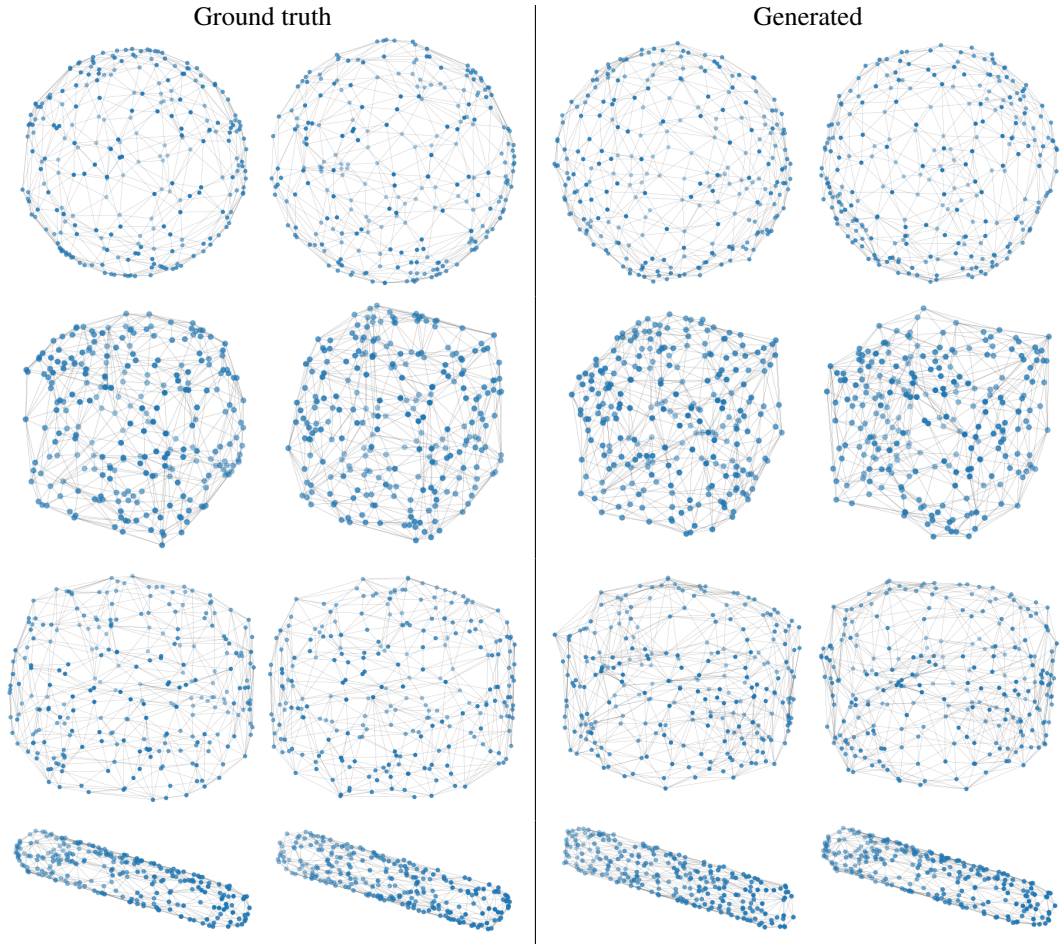


Figure 12: Training set (left) and generated objects (right) for different shapes (sphere, cubic box, cylinder, capsule).

Table 2: Hyperparameter optimisation scope for Sudoku experiments.

Transformer d_{model}	Transformer h	MLP d_{hidden}	MLP N	Learning rate α_{ref}	Reference guidance ω
$\{128, 256\}$	$\{2, 4\}$	$\{128, 512, 1024\}$	$\{2, 4, 6, 8\}$	$[10^{-4}, 5 \times 10^{-3}]$	$(1, 5]$

and cosine decay until 100,000 iterations. The maximum learning rate is denoted by α_{ref}. We use the AdamW optimiser. Gradient norm clipping is also applied.

Hyperparameter optimisation (HPO). We used Optuna [1] to conduct a Bayesian HPO study of training configurations for puzzles with 31–42 clues. Table 2 shows the HPO search space. All experiments in Section 6 use the following hyperparameters: $d_{model} = 256, h = 2, d_{hidden} = 128, N = 4, \alpha_{ref} = 6 \times 10^{-4}$, and $\omega = 5.0$.

NFE Ablation. At inference time, our model performs 25 decoding steps to generate a complete Sudoku solution from random noise. We investigate how the number of function evaluations (NFE) affects solving performance across clue counts, and whether the trained model can achieve one- or few-step generation.

Table 3 summarises the generation performance under varying NFEs. The results show that the proposed model is highly effective in the low-NFE regime for in-distribution puzzles (31 and 42 clues). Remarkably, a single network evaluation already achieves a success rate above 90% in both

Table 3: Success rate (%) vs. number of function evaluations (NFE) across clue counts. Experiments were run with five seeds, 200 samples per test.

	Harder OOD			In-distribution	
NFE	**17 clues**	**23 clues**	**29 clues**	**31 clues**	**42 clues**
1	5.9 ± 3.4	55.1 ± 16.5	89.2 ± 7.3	93.7 ± 6.2	99.7 ± 0.4
2	34.7 ± 9.2	84.3 ± 8.1	98.0 ± 1.6	99.1 ± 0.7	100.0 ± 0.0
3	37.4 ± 7.7	86.6 ± 5.7	98.7 ± 1.1	99.3 ± 0.5	99.9 ± 0.2
5	51.7 ± 7.7	94.3 ± 3.4	98.2 ± 1.9	99.7 ± 0.4	99.8 ± 0.2
10	59.9 ± 9.4	94.1 ± 4.2	99.3 ± 1.2	99.7 ± 0.4	100.0 ± 0.0
25	66.5 ± 10.0	95.9 ± 3.3	99.3 ± 1.2	100.0 ± 0.0	100.0 ± 0.0

settings, indicating that the learned denoising trajectory is highly structured and can often map noise directly to valid solutions in one step. Increasing the NFE to 2 further boosts performance to 99.1% and 100%, respectively, effectively reaching saturated performance with negligible additional computation. Near-OOD puzzles (29 clues) exhibit similar behaviour: one-step generation already achieves $89.2\% \pm 7.3\%$, while NFE $= 2$ is sufficient to achieve approximately 98% success. These findings demonstrate that the model behaves as a near-one-step solver on in-distribution and mildly shifted distributions, achieving strong accuracy with extremely limited iterative refinement.

The harder OOD regimes benefit more from additional integration steps. For the challenging 17- and 23-clue puzzles, success rates increase from 5.9% and 55.1% at NFE $= 1$ to 59.9% and 94.1% at NFE $= 10$, respectively, with marginal returns thereafter (66.5% and 95.9% at NFE $= 25$). The model degrades as clue count decreases, while retaining strong performance under moderate NFEs. Overall, the results highlight that the proposed flow-matching formulation learns highly efficient generation dynamics, enabling near-one-step inference for in-distribution Sudoku solving while remaining robust under progressively harder distribution shifts.

E.4 Molecule generation

We evaluate unconditional molecular graph generation on QM9 [41], generating both atom identities and bond connectivity. The molecular graph defining the training heat operator is unavailable at inference time. Molecules contain at most nine heavy atoms of types C, N, O, and F, with implicit hydrogens and kekulised single, double, or triple bonds. The data split contains 97,734 training, 20,042 validation, and 13,055 test graphs.

E.4.1 Setup

Continuous molecular representation. For a molecule with n heavy atoms (with $n \leq 9$), let $\mathbf{N} \in \{0,1\}^{n\times 4}$ contain one-hot atom labels, with columns corresponding to C, N, O, and F. Let $\mathbf{A} \in \{0,1,2,3\}^{n\times n}$ be the symmetric bond-order adjacency matrix with zero diagonal, extending the adjacency convention in Section 2 to integer weights. With $\mathbf{D}$ the diagonal matrix of weighted degrees, we write the eigendecomposition of the (symmetric positive semidefinite) bond Laplacian and define the factor:

$$\mathbf{L}_{\text{bond}} := \mathbf{D} - \mathbf{A} = \mathbf{V}_{\text{bond}}\mathbf{\Lambda}_{\text{bond}}\mathbf{V}_{\text{bond}}^{\top}, \qquad \mathbf{Z} := \mathbf{V}_{\text{bond}}\mathbf{\Lambda}_{\text{bond}}^{1/2}. \tag{88}$$

$\mathbf{Z}$ in Eq. (88) is an $n \times n$ matrix; we append $9 - n$ zero columns to obtain $\mathbf{Z} \in \mathbb{R}^{n\times 9}$ (see details in Appendix E.4.6). Together with the four atom-label channels, this gives the clean state $\bar{\mathbf{X}}_1 = [\mathbf{N}, \mathbf{Z}] \in \mathbb{R}^{n\times 13}$. Following the paper's reverse-time convention, $\bar{\mathbf{X}}_0$ is noise and $\bar{\mathbf{X}}_1$ is clean data. The heat transfer operator is parameterised by the following matrix:

$$\mathbf{L} = \mathbf{L}_{\gamma,\delta} = -\gamma\left(\frac{\mathbf{L}_{\text{bond}}}{n} + \delta\mathbf{I}\right), \qquad \gamma = 100, \quad \delta = 0.05. \tag{89}$$

Generation starts from independent standard Gaussian entries, with n sampled from the empirical training atom-count distribution. Atom types are decoded by $\arg\max$, and bond orders are obtained by rounding $-(\widehat{\mathbf{Z}}\widehat{\mathbf{Z}}^{\top})_{ij}$ to the nearest element of $\{0,1,2,3\}$, with zero diagonal (see Eq. (91)), where $\widehat{\mathbf{Z}} \in \mathbb{R}^{n\times 9}$ is the generated matrix. The decoder requires no learned bond predictor.

Table 4: Unconditional molecular graph generation on QM9. Values are mean ± sample standard deviation across three training seeds, with 10,000 raw draws per seed. The first five metrics are percentages. Connected validity requires an intact, connected, neutral-valid molecule; validity uses the charge-aware valence check. Yield counts distinct novel connected-valid molecules per raw draw.

Method	Conn. valid ↑	Valid ↑	Unique ↑	Novel ↑	Yield ↑	FCD ↓	NSPDK ↓
GDSS [21]	73.90 ± 1.72	96.42 ± 1.32	99.09 ± 0.12	78.23 ± 1.12	57.30 ± 2.12	2.405 ± 0.169	0.003496 ± 0.000972
GruM [22]	99.49 ± 0.09	99.84 ± 0.03	96.77 ± 0.36	22.19 ± 0.37	21.37 ± 0.36	0.085 ± 0.002	0.000192 ± 0.000011
DiGress [51]	98.99 ± 0.12	99.22 ± 0.09	96.24 ± 0.28	33.47 ± 0.20	31.89 ± 0.26	0.230 ± 0.017	0.000391 ± 0.000012
CatFlow [15]	99.05 ± 0.09	99.78 ± 0.03	96.97 ± 0.11	24.92 ± 0.38	23.93 ± 0.36	0.157 ± 0.013	0.000316 ± 0.000015
Standard FM [15]	98.13 ± 0.09	99.35 ± 0.09	96.99 ± 0.11	29.64 ± 0.09	28.21 ± 0.10	0.261 ± 0.024	0.000405 ± 0.000048
SHDM (Ours)	90.90 ± 0.56	98.48 ± 0.39	98.33 ± 0.15	75.09 ± 0.72	67.12 ± 0.38	2.574 ± 0.131	0.004762 ± 0.000432

Denoiser and objective. We use a node/pair/global Transformer inspired by DiGress [51] and GruM [22], with eight blocks, eight attention heads, node width 256, and approximately 5.55 million parameters. Following Section 4.2, the network predicts the average velocity over each interval (a, b); its output is converted to a next-state prediction via $\widehat{\bar{\mathbf{X}}}_b = \bar{\mathbf{X}}_a + (b-a)\,\mathrm{net}_\theta(\bar{\mathbf{X}}_a, a, b)$. Training combines atom-weighted regression on this predicted next state, endpoint atom regression through a short rollout, off-diagonal Gram reconstruction, and an excess-valence penalty:

$$\mathcal{L}_{\mathrm{QM9}} = \mathcal{L}_{\mathrm{step}} + 0.25\mathcal{L}_{\mathrm{roll,atom}} + 0.025\mathcal{L}_{\mathrm{Gram}} + 0.01\mathcal{L}_{\mathrm{val}}. \tag{90}$$

Atom channels receive twice the per-coordinate regression weight of factor channels. Gradients pass through the final three velocity-based state updates, and the Gram and valence terms supervise the resulting endpoint. Section E.4.3 defines the loss terms. Training and sampling use the exponential schedule in Eq. (8), with $T = 32$ and $\kappa = 5$.

Evaluation. We trained three independent SHDM models for 1,000 epochs each and generated 10,000 raw draws from each final EMA checkpoint. We report connected neutral validity, charge-aware valence validity, uniqueness, novelty, distinct novel connected-valid yield, Fréchet ChemNet Distance (FCD) [40], and NSPDK [11]. We compare with local reruns of GDSS, GruM, DiGress, CatFlow, and standard flow matching. Novelty is measured against the training set. Table 4 reports means and sample standard deviations across training seeds.

Results. SHDM achieves connected neutral validity of $(90.90 \pm 0.56)\%$, valence validity of $(98.48 \pm 0.39)\%$, and a distinct novel connected-valid yield of $(67.12 \pm 0.38)\%$. Conditional uniqueness and novelty are $(98.33 \pm 0.15)\%$ and $(75.09 \pm 0.72)\%$, respectively. FCD [40] is 2.574 ± 0.131 and NSPDK [11] is 0.004762 ± 0.000432. SHDM has the highest distinct novel connected-valid yield among the reported methods, although its larger distributional distances indicate weaker agreement with the QM9 test distribution. Figure 13 shows examples decoded from the generated continuous states.

E.4.2 Implementation details on generation

Representation, training paths, and decoding. The factorisation in Eq. (88) uses the full, unnormalised, bond-weighted Laplacian, so $\mathbf{Z}\mathbf{Z}^\top = \mathbf{L}_{\mathrm{bond}}$. Factor columns are padded to nine and remain part of the regression target; padded node rows are masked. The heat operator uses each molecule's actual atom count n and acts along the node dimension of every state channel. Eigenvector signs are fixed using weighted-degree alignment; when this alignment is numerically zero, the largest-magnitude entry is made positive. As discussed in Appendix D, this convention does not resolve rotations within repeated eigenspaces.

Training pairs follow the conditional flow in Eq. (7). For each graph, we uniformly sample an adjacent interval $(a, b) = (\tau_i, \tau_{i+1})$ from the exponential grid. Gaussian noise and interval indices are resampled at each update. The network receives the noisy state and time channels $(a, b-a)$ and predicts the average velocity; its output is converted to a next-state prediction via $\widehat{\bar{\mathbf{X}}}_b = \bar{\mathbf{X}}_a + (b-a)\,\mathrm{net}_\theta(\bar{\mathbf{X}}_a, a, b)$.

Training paths and unconditional generation start from independent standard Gaussian entries. With $\gamma\delta = 5$, the slowest Laplacian mode, whose eigenvalue is $-\gamma\delta = -5$, retains approximately 0.67% of its clean amplitude after unit noising time, with noise variance approximately 0.999955; Gaussian initialisation is therefore a finite-time approximation.

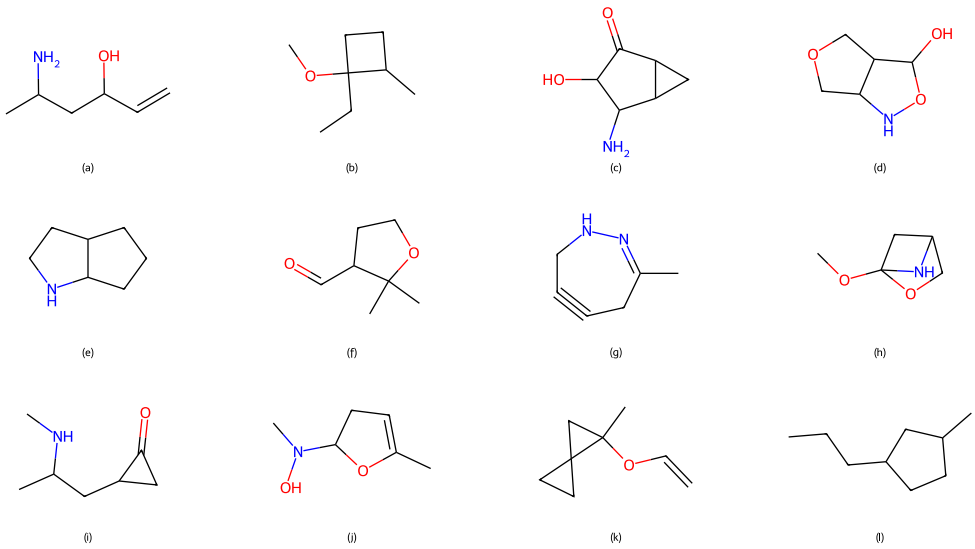


Figure 13: Examples of molecules generated by SHDM on QM9. Twelve generated samples were selected uniformly at random. Drawings show generated atom types and bond connectivity with implicit hydrogens, without bond repair or fragment extraction.

At inference time, our generation process samples n from the empirical training atom-count distribution and applies the 32 scheduled updates without access to the clean graph or the eigendecomposition of the Laplacian. Atom labels are decoded by argmax, and bond orders by

$$\forall i \neq j,\ \widehat{\mathbf{A}}_{ij} = \operatorname*{arg\,min}_{r\in\{0,1,2,3\}} \left(-(\widehat{\mathbf{Z}}\widehat{\mathbf{Z}}^{\top})_{ij} - r\right)^2, \qquad \widehat{\mathbf{A}}_{ii} = 0, \tag{91}$$

where $\widehat{\mathbf{Z}} \in \mathbb{R}^{n\times 9}$ is the generated matrix. The decoded adjacency is symmetric, but chemical validity is not guaranteed.

Architecture. The eight-block Transformer uses eight attention heads and node/pair/global widths of 256/64/64. Feed-forward widths are 512/128/128, with ReLU activations, residual connections, and layer normalisation. We use no dropout or positional encoding.

At the current state $\bar{\mathbf{X}}_a$, let $\widehat{a}_i$ be the atom type decoded by argmax and let $\widehat{\mathbf{A}}^{(a)}$ be the bond-order matrix decoded using Eq. (91). Define the current decoded bond valence as $v_i^{\text{current}} := \sum_{j\neq i} \widehat{\mathbf{A}}_{ij}^{(a)}$. The reference-valence function maps C, N, O, and F to $(4, 3, 2, 1)$, respectively, and the signed reference-valence deficit is

$$d_i^{\text{ref}} := v_{\text{ref}}(\widehat{a}_i) - v_i^{\text{current}}. \tag{92}$$

Node inputs comprise the continuous state, time channels $(a, b-a)$, v_i^{current}, d_i^{ref}, and normalised atom count $n/9$. Pair inputs are derived from $-\mathbf{Z}_a\mathbf{Z}_a^{\top}$, and global inputs are $(a, b-a, n/9)$. Pair updates are symmetrised. A width-128 MLP combines node, pooled pair, and global representations to predict 13 average velocity channels; the resulting output is converted into the next-state prediction.

E.4.3 Training loss

Let $m_{\mathcal{B}}$ be the total number of real atoms in a minibatch. State norms sum over real node rows and all factor columns. For a predicted state $\mathbf{X}$ and its target $\mathbf{Y}$, subscripts N and Z denote their atom-label and factor-coordinate blocks, respectively. For states $\mathbf{X} = [\mathbf{X}_N, \mathbf{X}_Z]$ and $\mathbf{Y} = [\mathbf{Y}_N, \mathbf{Y}_Z]$, define

$$\mathcal{E}_{\text{state}}(\mathbf{X}, \mathbf{Y}) := \frac{2\|\mathbf{X}_N - \mathbf{Y}_N\|_F^2 + \|\mathbf{X}_Z - \mathbf{Y}_Z\|_F^2}{13 m_{\mathcal{B}}}. \tag{93}$$

The factor 2 weights atom regression, and 13 is the total number of atom and factor channels. The transition loss is $\mathcal{L}_{\text{step}} = \mathbb{E}[\mathcal{E}_{\text{state}}(\widehat{\mathbf{X}}_b, \bar{\mathbf{X}}_b)]$, without residual-dependent normalisation ($p = 0$ in Algorithm 1). Expectations are over minibatches, noise, and sampled intervals.

Endpoint atom rollout. We use differentiable rollout supervision to account for errors across successive predictions, related to multi-step training with differentiable solvers [49]. We use an independent Gaussian draw to construct a conditional state at τ_{29}. Three network updates produce the endpoint $\widetilde{\widetilde{\mathbf{X}}}_1 = [\widetilde{\mathbf{N}}, \widetilde{\mathbf{Z}}]$, with gradients passing through all three updates. We supervise endpoint atoms using the following equation:

$$\mathcal{L}_{\mathrm{roll,atom}} = \mathbb{E}\left[\frac{2\|\widetilde{\mathbf{N}} - \mathbf{N}\|_F^2}{13 m_{\mathcal{B}}}\right]. \tag{94}$$

Balanced Gram reconstruction. We supervise the bond-Laplacian Gram matrix with balanced squared error, related to inner-product reconstruction and edge/nonedge reweighting in graph autoencoders [24]. Let $\mathcal{P}_+$ and $\mathcal{P}_0$ contain unordered off-diagonal pairs within individual molecules with nonzero and zero clean bond order, respectively, and let $\mathcal{H}$ contain the nonempty groups. We define

$$\mathcal{L}_{\mathrm{Gram}} := \mathbb{E}\left[\begin{cases} \dfrac{1}{|\mathcal{H}|}\displaystyle\sum_{\mathcal{P}\in\mathcal{H}} \frac{1}{|\mathcal{P}|}\sum_{(i,j)\in\mathcal{P}} \Big((\widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\top)_{ij} - (\mathbf{Z}\mathbf{Z}^\top)_{ij}\Big)^2, & |\mathcal{H}| > 0, \\ 0, & |\mathcal{H}| = 0 \end{cases}\right]. \tag{95}$$

Bonded and nonbonded groups receive equal total weight when both are present. Padding, diagonal entries, and pairs across molecules are excluded; the loss is zero when no eligible pairs exist.

Excess-valence penalty. We use a differentiable excess-valence penalty, related to soft valence constraints for molecular graph generation [35]. At the predicted endpoint, define soft bond probabilities

$$p_{ij}(r) := \frac{\exp[-10(-(\widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\top)_{ij} - r)^2]}{\sum_{s=0}^{3}\exp[-10(-(\widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\top)_{ij} - s)^2]}, \qquad r \in \{0,1,2,3\}, \quad i \neq j. \tag{96}$$

The logit scale 10 is fixed. Let $\widetilde{v}_i = \sum_{j\neq i}\sum_{r=0}^{3} r\, p_{ij}(r)$ and $c_i = \max\{v_{\mathrm{ref}}(a_i), \sum_j \mathbf{A}_{ij}\}$, where a_i is the clean atom type and the reference valences for C, N, O, and F are $(4,3,2,1)$. We penalise excess valence using

$$\mathcal{L}_{\mathrm{val}} = \mathbb{E}\left[\frac{1}{m_{\mathcal{B}}}\sum_{i\ \mathrm{real}} \max(0, \widetilde{v}_i - c_i)^2\right]. \tag{97}$$

These auxiliary losses supervise training; no repair is applied during generation.

E.4.4 Optimisation and hardware

We trained three independently initialised models for 1,000 epochs each using the settings in Table 5. The configuration was fixed before final evaluation.

Table 5: Training and sampling hyperparameters for SHDM on QM9.

Hyperparameter	Value
Heat scale γ; shift δ	100; 0.05
Training and sampling grid	$T = 32$, exponential $\kappa = 5$
Endpoint rollout	3 updates
Optimiser	AdamW
Learning rate; weight decay	3×10^{-4} (constant); 0.01
Gradient-norm clipping	1
EMA decay	0.999
Training; generation batch sizes	32; 128
Precision	float32, TF32 disabled

Training used an NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition GPU (96 GB). Final evaluation used the fixed epoch-1,000 EMA checkpoint.

E.4.5 Metrics

All metrics use unmodified raw draws and a shared RDKit [42] evaluator. Connected validity is the proportion of draws yielding intact, connected molecules that pass neutral RDKit sanitisation. Valence validity is the proportion passing the charge-aware GruM check, without requiring connectivity. Uniqueness is the fraction of connected-valid draws that are distinct; novelty is the fraction of these distinct molecules absent from the model's actual training population. Yield is the number of distinct novel connected-valid molecules divided by the total number of draws.

FCD [40] compares ChemNet feature distributions of connected, neutral-valid generated molecules and test-set molecules, retaining duplicates. NSPDK [11, 21] uses all raw graphs, including invalid and duplicate outputs, and reports biased squared MMD with the GDSS complexity-4 features and linear kernel.

Metrics are computed separately for each training seed and reported as mean and sample standard deviation across three independently trained models, with 10,000 raw draws per model.

E.4.6 Padding Explanation

In this experiment, $\mathbf{Z}$ is an exact factor of $\mathbf{L}_{\text{bond}}$ (i.e., $\mathbf{L}_{\text{bond}} = \mathbf{Z}\mathbf{Z}^\top$). Here, n counts the heavy atoms in one molecule, so $n \leq 9$ for QM9. Hydrogens are implicit.

Let the unpadded factor be

$$\mathbf{Z}_{\text{raw}} = \mathbf{V}_{\text{bond}}\mathbf{\Lambda}_{\text{bond}}^{1/2} \in \mathbb{R}^{n\times n}.$$

Zero-padding is straightforward: we append $9-n$ zero columns,

$$\mathbf{Z} = \begin{bmatrix}\mathbf{Z}_{\text{raw}} & \mathbf{0}_{n\times(9-n)}\end{bmatrix} \in \mathbb{R}^{n\times 9}.$$

For example, a 7×7 factor becomes 7×9.

The appended columns do not affect the reconstruction:

$$\begin{aligned}\mathbf{Z}\mathbf{Z}^\top &= [\mathbf{Z}_{\text{raw}} \;\; \mathbf{0}]\begin{bmatrix}\mathbf{Z}_{\text{raw}}^\top \\ \mathbf{0}\end{bmatrix} \\ &= \mathbf{Z}_{\text{raw}}\mathbf{Z}_{\text{raw}}^\top \\ &= \mathbf{V}_{\text{bond}}\mathbf{\Lambda}_{\text{bond}}\mathbf{V}_{\text{bond}}^\top = \mathbf{L}_{\text{bond}}.\end{aligned}$$

For a connected molecule, both $\mathbf{Z}$ and $\mathbf{L}_{\text{bond}}$ have rank at most $n-1 \leq 8$. No nonzero spectral information is discarded.

For bond reconstruction, the key identity is $\mathbf{L}_{\text{bond}} = \mathbf{D} - \mathbf{A}$ where $\mathbf{D}$ is a diagonal matrix: the off-diagonal entries of $\mathbf{L}_{\text{bond}}$ are the negative bond orders. Therefore,

$$\forall i \neq j,\; \mathbf{A}_{ij} = -(\mathbf{Z}\mathbf{Z}^\top)_{ij}.$$

This identity is exact for the clean factor. For a generated factor $\widehat{\mathbf{Z}}$, we instead round $-(\widehat{\mathbf{Z}}\widehat{\mathbf{Z}}^\top)_{ij}$ to the nearest allowed bond order in $\{0, 1, 2, 3\}$ and set the diagonal to zero.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We propose a new type of diffusion model that is based on the heat equation. All our theoretical claims are justified, either via references with precise chapters or with proofs in the appendix.

2. **Limitations**

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

   Answer: [Yes]

   Justification: See Section 8.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: We propose a new type of diffusion model that is based on the heat equation. All our theoretical claims are justified, either via references with precise chapters or with proofs in the appendix.

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We provide details to reproduce our experiments.

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

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

   Answer: [No]

   Justification: We provide the equations to calculate the average velocity. The rest can be easily implemented with any dataset and neural network architecture.

6. **Experimental setting/details**

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

   Answer: [No]

   Justification: We provide a lot of details in the main paper and Appendix E about the choice of hyperparameters.

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: See Table 3 and Fig. 6.

8. **Experiments compute resources**

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

   Answer: [Yes]

Justification: See Appendix E.

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have done so.

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: We see no societal impacts as our work is theoretical. See Section 9.

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We cite all the references related to the assets and datasets that we use.

13. **New assets**

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

    Answer: [No]

    Justification: No new assets were proposed, only a new algorithm.

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs.